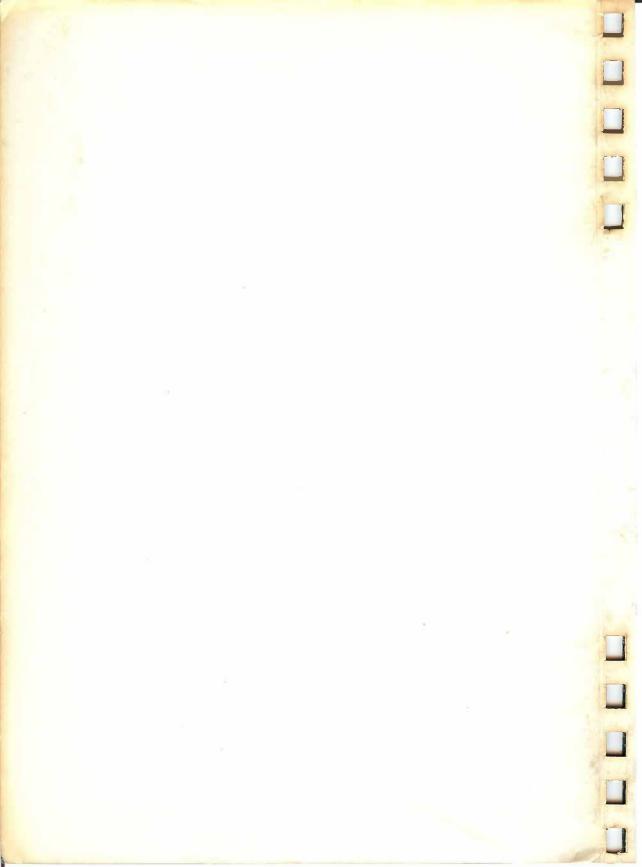
## COMPUTE!'s SECOND BOOK

More than 30 outstanding games, applications, tutorials, and utilities from COMPUTE! Publications. Animate graphics, explore dangerous dungeons, view the heavens, keep files, learn all about batch files, and much more. For users of the IBM PC, PC XT, and PCjr personal computers.

A COMPUTE! Books Publication

UTEI

\$14.95



# **COMPUTE!'s**

### Publications,Inc. Part of ABC Consumer Magazines, Inc. One of the ABC Publishing Companies

Greensboro, North Carolina

The following articles were originally published in COMPUTE! magazine, copyright 1984, COMPUTE! Publications, Inc.:

"Jackpot" (August); "Lightning Sort" (September),

The following articles were originally published in COMPUTE!'s PC & PCjr magazine, copyright 1984, COMPUTE! Publications, Inc.: "Color Directory" (March); "Simple Animation" (March); "Type Bomb" (April); "Dialing For Dollars: Writing Your Own Telecommunications Program" (May); "Rotating Graphics" (June); "Jigsaw" (July); "IBM BASIC Tips and Techniques" (September); "First Steps in Three-Dimensional Graphics" (October).

The following articles were originally published in COMPUTE! magazine, copyright 1985, COMPUTE! Publications, Inc.:

"Pie Chart Maker" (January); "IBM Screen Swapping" (February); "Bearmath" (March); "Webster Dines Out" (June); "Fast Filer" (July); "Animator" (August); "Filecopy" (August); "All About Batch Files" (September, October); "The Witching Hour" (October); "Simple Assembling with IBM DEBUG" (November); "Skyscape" (November); "Balloon Crazy" (December); "Memo Diary" (December); "Million-Color Palette" (December).

The following articles were originally published in COMPUTE! magazine, copyright 1986, COMPUTE! Publications, Inc.:

"IBM Advanced Function Key Techniques" (January); "Fractal Graphics" (March); "Switchbox" (March); "IBM Variable Snapshot" (April); "Screen Clock for IBM" (April); "Smooth-Scrolling Billboards" (April); "Hickory, Dickory, Dock" (May).

Copyright 1986, COMPUTE! Publications, Inc. All rights reserved.

Reproduction or translation of any part of this work beyond that permitted by Sections 107 and 108 of the United States Copyright Act without the permission of the copyright owner is unlawful.

Printed in the United States of America

10987654321

ISBN 0-87455-046-7

The authors and publisher have made every effort in the preparation of this book to insure the accuracy of the programs and information. However, the information and programs in this book are sold without warranty, either express or implied. Neither the authors nor COMPUTE! Publications, Inc., will be liable for any damages caused or alleged to be caused directly, indirectly, incidentally, or consequentially by the programs or information in this book.

The opinions expressed in this book are solely those of the authors and are not necessarily those of COMPUTE! Publications, Inc.

COMPUTE! Publications, Inc., Post Office Box 5406, Greensboro, NC 27403, (919) 275-9809, is part of ABC Consumer Magazines, Inc., one of the ABC Publishing Companies, and is not associated with any manufacturer of personal computers. IBM PC, PC XT, and PCjr are trademarks of International Business Machines, Inc.

### **Contents**

Foreword v
Chapter 1. Applications 1
Color Directory Charles Brannon
Memo Diary
Jim Butterfield 11
Fast Filer Richard Mansfield and Patrick Parrish
Screen Clock Marc Sugiyama
Smooth-Scrolling Billboards
Paul W. Carlson
Pie Chart Maker
Michael Posner
Chapter 2. Having Fun
Switchbox  Total University / PC / PC in previous by Time Visiters  E1
Todd Heimarck / PC/PCjr version by Tim Victor 51 Webster Dines Out
Walter Bulawa / PC/PCjr version by Charles Brannon 62
The Witching Hour
Brian Flynn
Jackpot
Rick Rothstein / IBM PC/PCjr version by Kevin Mykytyn 72
Balloon Crazy
Joseph Russ / IBM PC/PCjr version by Charles Brannon 78
Jigsaw
David Bohlke
Deadly Dungeon  Jeff and John Klein
Jeff and John Klein89
Chapter 3. Education
Type Bomb Clark and Kathy Kidd
Bearmath
Gary West and Jim Bryan 111
Skyscape 122
Robert M. Simons / IBM PC/PCjr version by Tim Victor 122
Hickory, Dickory, Dock Barbara H. Schulak / IBM PC/PCjr version by Tim Victor 135

Chapter 4. Graphics	139
Steve Johnson	141
Million-Color Palette  John and Jeff Klein	157
Rotating Graphics  Michael A. Covington	170
First Steps in Three-Dimensional Graphics	
Michael A. Covington	176
Fractal Graphics Paul W. Carlson	186
Simple Animation	
Joseph Juhasz	193
Chapter 5. All About Programming	199
Simple Assembling with IBM DEBUG	1,,,
Tim Victor	201
All About Batch Files G. Russ Davies	209
Lightning Sort	207
Russ Gaspard / PC/PCjr version by Tim Victor	226
Dialing for Dollars:	
Dialing for Donars.	
Writing Your Own Telecommunications Program	220
Writing Your Own Telecommunications Program  Donald B. Trivette	230
Writing Your Own Telecommunications Program	
Writing Your Own Telecommunications Program Donald B. Trivette	250
Writing Your Own Telecommunications Program Donald B. Trivette	
Writing Your Own Telecommunications Program Donald B. Trivette	<ul><li>250</li><li>255</li></ul>
Writing Your Own Telecommunications Program Donald B. Trivette	<ul><li>250</li><li>255</li><li>264</li></ul>
Writing Your Own Telecommunications Program Donald B. Trivette Advanced Function Key Techniques Peter F. Nicholson, Jr.  Variable Snapshot Tony Roberts IBM BASIC Tips and Techniques C. Regena Screen Swapping Paul W. Carlson	<ul><li>250</li><li>255</li><li>264</li></ul>
Writing Your Own Telecommunications Program Donald B. Trivette  Advanced Function Key Techniques Peter F. Nicholson, Jr.  Variable Snapshot Tony Roberts  IBM BASIC Tips and Techniques C. Regena  Screen Swapping Paul W. Carlson  Filecopy	<ul><li>250</li><li>255</li><li>264</li><li>269</li></ul>
Writing Your Own Telecommunications Program Donald B. Trivette Advanced Function Key Techniques Peter F. Nicholson, Jr.  Variable Snapshot Tony Roberts IBM BASIC Tips and Techniques C. Regena Screen Swapping Paul W. Carlson	<ul><li>250</li><li>255</li><li>264</li><li>269</li></ul>
Writing Your Own Telecommunications Program Donald B. Trivette  Advanced Function Key Techniques Peter F. Nicholson, Jr.  Variable Snapshot Tony Roberts  IBM BASIC Tips and Techniques C. Regena  Screen Swapping Paul W. Carlson  Filecopy	<ul><li>250</li><li>255</li><li>264</li><li>269</li></ul>
Writing Your Own Telecommunications Program Donald B. Trivette  Advanced Function Key Techniques Peter F. Nicholson, Jr.  Variable Snapshot Tony Roberts  IBM BASIC Tips and Techniques C. Regena Screen Swapping Paul W. Carlson  Filecopy John and Jeff Klein  Appendices A. Beginner's Guide to Typing In Programs	<ul><li>250</li><li>255</li><li>264</li><li>269</li></ul>
Writing Your Own Telecommunications Program Donald B. Trivette  Advanced Function Key Techniques Peter F. Nicholson, Jr.  Variable Snapshot Tony Roberts  IBM BASIC Tips and Techniques C. Regena  Screen Swapping Paul W. Carlson  Filecopy John and Jeff Klein  Appendices A. Beginner's Guide to Typing In Programs B. The Automatic Proofreader	<ul><li>250</li><li>255</li><li>264</li><li>269</li><li>274</li><li>279</li></ul>
Writing Your Own Telecommunications Program Donald B. Trivette  Advanced Function Key Techniques Peter F. Nicholson, Jr.  Variable Snapshot Tony Roberts  IBM BASIC Tips and Techniques C. Regena  Screen Swapping Paul W. Carlson  Filecopy John and Jeff Klein  Appendices A. Beginner's Guide to Typing In Programs B. The Automatic Proofreader Charles Brannon	<ul><li>250</li><li>255</li><li>264</li><li>269</li><li>274</li><li>279</li><li>282</li></ul>
Writing Your Own Telecommunications Program Donald B. Trivette  Advanced Function Key Techniques Peter F. Nicholson, Jr.  Variable Snapshot Tony Roberts  IBM BASIC Tips and Techniques C. Regena  Screen Swapping Paul W. Carlson  Filecopy John and Jeff Klein  Appendices A. Beginner's Guide to Typing In Programs B. The Automatic Proofreader	<ul><li>250</li><li>255</li><li>264</li><li>269</li><li>274</li><li>279</li></ul>
Writing Your Own Telecommunications Program Donald B. Trivette  Advanced Function Key Techniques Peter F. Nicholson, Jr.  Variable Snapshot Tony Roberts  IBM BASIC Tips and Techniques C. Regena  Screen Swapping Paul W. Carlson  Filecopy John and Jeff Klein  Appendices A. Beginner's Guide to Typing In Programs B. The Automatic Proofreader Charles Brannon	<ul><li>250</li><li>255</li><li>264</li><li>269</li><li>274</li><li>279</li><li>282</li></ul>

### **Foreword**

COMPUTE! Publications has been supporting the IBM PC and PCjr for more than three years, and has published a wideranging library of easy-to-understand articles and dazzling software. COMPUTE!'s Second Book of IBM pulls together the best we've published and puts it in one volume, all here at your fingertips.

You're holding an impressive collection of computer software in your hands. More than 30 programs—all ready to type in and use—all for the IBM PC, PCjr, and PC-compatible

computers.

And like its predecessor, COMPUTE!'s First Book of IBM, this anthology offers more than just high quality, inexpensive software. There are informative tutorials that show you how to do everything from displaying over a million colors on your screen to automating almost any computer activity with powerful batch files.

Practical applications like "Memo Diary" and "Fast Filer" let you write and file with ease. Entertaining games put you in damp dungeons and under a deadly barrage of balloons. Educational programs, like "Hickory, Dickory, Dock" and "Skyscape" teach time and display the night sky from anywhere in the world.

Graphics programs and tutorials, always impressive, demonstrate fractals, rotating pictures, and three-dimensional drawings. There's even a sophisticated frame-by-frame animator which lets you turn your PC or PCjr into a miniature cartoon studio. And programming utilities and information—from fast file copying to a telecommunications program that automatically retrieves stock quotations—put more punch in your PC.

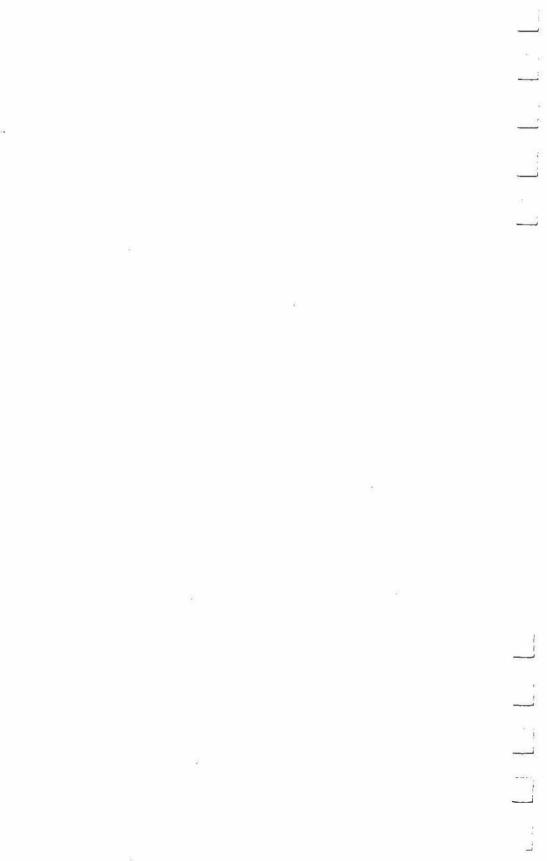
It's all here. Best of all, it's all understandable. No unnecessary technical jargon to wade through. Clearly written by people who know the PC and PCjr, every article and program

has been thoroughly checked for accuracy.

To make it easy to type in the programs, we've included "The Automatic Proofreader," an error-checking aid which makes sure you enter each program correctly the *first time*. But if you prefer not to type in the programs, you can order the companion disk for *COMPUTE!'s Second Book of IBM*. All the

programs in this book are available on a disk which you can run on your PC, PCjr; or PC-compatible computer. Look for the coupon in the back of this book for more details, or call toll-free 1-800-346-6767 (in NY, 1-212-887-8525).

### CHAPTER ONE Applications



### **Color Directory**

Charles Brannon

"Color Directory," for the PC and PCjr, helps you manage your disks. It overcomes the 11-character filename limit of PC-DOS by letting you add 24-character descriptions. It runs on any PCjr or PC with color/graphics board and DOS 2.0 or higher.

With PC-DOS (MS-DOS), IBM chose to carry over many of the conventions of an earlier operating system called CP/M. These include a similar command structure, batch processing, .COM files for machine language programs, and filename conventions. While PC-DOS borrowed the best of CP/M and improved upon it, there are a few undesirable vestiges.

When you need to store information on disk, it's accessed via a *filename*, a series of letters and numbers, which should be descriptive. In CP/M and PC-DOS, a filename can be eight characters long, with an optional three-character *extension*.

Sometimes the extension is filled in for you. If you save a BASIC program to disk under the name QUIKSORT, for instance, it will be listed as QUIKSORT.BAS. BASIC added the extension. You can see this by typing the command FILES from BASIC, or DIR from DOS. The eight-character limit forced us to "creatively" spell Quicksort as QUIKSORT.

### **Filename Limitations**

The rather short filename length can be quite a limitation. You can probably figure out what QUIKSORT stands for, but how do you cope with names like EXEC2.BIN, INDPTR.COM, or CREBASE.BAT? Many DOS's permit much longer names. A Commodore filename can be 16 characters long; an Apple filename, 30 characters. As an extreme, the Epson QX-10 lets you use a long phrase to describe a filename, even sentences like *The table of contents for the January issue*.

"Color Directory" lets you document all your files in the same fashion. You can link a 24-character description to any or all of the files on a disk. Every time you run Color Directory, it displays the directory and the descriptions. You can enter or change a description at any time. You can even enter

a descriptive name for the disk itself, such as PC DOS System Master.

In addition, Color Directory lets you run any BASIC program just by pressing a function key, then the Enter key. Color Directory displays up to eight files at a time. If you have more than eight files on your disk, you can press PgDn to see the next page. You can also press PgUp to page backward.

### The Selected Filename Blinks

When you press a function key, the associated filename blinks to show it's selected. You're then given three choices:

Press Enter to run the program

• Press the space bar to enter the description

 Press the Esc (Escape) key to return to the menu (You'd press Esc if you change your mind and want to select a different file.)

If you press the space bar, the existing description is shown at the bottom of the screen. If you haven't yet named the disk, you'll be prompted to name it before entering descriptions for the filenames. You can then edit, enter, or change the description for the file you've selected.

You can also run any BASIC program if you press Enter after you've selected a file with one of the function keys. Run only those programs which are actually in BASIC (these files usually have an extension of .BAS). If the file is *not* a BASIC program, you may see the message *Direct Statement in File* and find that Color Directory has disappeared.

The Description File

Before the selected program is run, the descriptions are saved to disk with the filename DESCR.DIR. If there's an error while Color Directory is writing (which can occur if the disk is full, the write-protect tab is on, the drive door is open, and so on), you'll see the message Cannot save descriptions. Run program anyway? (Y/N). If you press Y, Color Directory will try to run the program without saving the descriptions. If you don't want to lose your descriptions, you should press N, then figure out what went wrong.

### Quit Menu

You can also save the descriptions without running a program. At the main menu (before you press a function key), press Esc. The Quit menu appears:

- 1. Exit to BASIC
- 2. Exit to DOS
- 3. Re-RUN
- 4. Save Descriptions
- 5. Menu

Press 4 to write the descriptions out to the disk. If there's an error, Color Directory will detect it and inform you. Press Enter to return to the main menu. You can also use the other options to exit or rerun Color Directory (Color Directory redefines the function keys, so if you exit to BASIC the keys will no longer perform their predefined functions). You can also return to the main menu (if you pressed Esc by mistake, perhaps).

If you have a printer attached and turned on, press Shift-PrtSc to dump a copy of the screen to the printer. If you set your printer to condensed mode with LPRINT CHR\$(15), the directory, with descriptions, is small enough to tape to the disk envelope.

You can use either drive A or drive B with Color Directory (PC only), and you'll be asked to press either A or B when you first run it. If you want to customize Color Directory for single-drive systems, remove the keyword REM from line 160, leaving the rest of the line in place. Color Directory will then read from drive A.

### **Automate Your PC**

To automate Color Directory even further, use COPY to move BASIC.COM to your disk (from the DOS System disk). You should also have DOS on the disk. Then enter this line from BASIC to create a batch file:

### OPEN "AUTOEXEC.BAT" FOR OUTPUT AS #1:PRINT #1,"BASIC COLORDIR":CLOSE#1

This assumes that you've saved Color Directory as COLORDIR. Now, every time you turn on your machine with the disk in drive A, Color Directory automatically runs and displays all

the files. You can do this to all your disks, adding an instant, easy-to-use menu program.

**Technical Tips** 

The BASIC command FILES can only display the directory—it doesn't process the directory information in other ways. You can make it selective by giving a filename with wildcards; for example, FILES "B:\*.BAS" will list all files on drive B: with an extension of .BAS (BASIC programs). But there's no way to use FILES that lets BASIC read and process the filenames. They're sitting on the screen, but BASIC can't use that. We need to read a disk directory into a string array, so a program such as Color Directory can use a statement such as RUN F\$(X) to execute the file, depending on X.

The trick is to display the directory on the screen, then use LINE INPUT to read each line of the directory. But LINE INPUT, like INPUT, requires that you press Enter after each line. If only IBM had permitted us to OPEN a file to the screen for INPUT. The solution was to make the PC "press" its own keys, by POKEing into the keyboard buffer.

The keyboard buffer is a 30-character block of memory at 41AH, segment 0 (in decimal, 1050). Two pointers, called HEAD and TAIL, keep track of the first and last keys in the buffer. The buffer is circular. The HEAD is not necessarily at 41AH. It can start in the middle, with TAIL wrapping around as more characters are saved in the buffer. The PC keyboard starts to beep if you try to type past the 15-character limit (when TAIL bumps into HEAD).

The buffer stores two bytes per key pressed. The first byte in the pair is the ASCII representation; the second is the keyboard scan code. To program the buffer, first set HEAD to 1050 (to make POKEing convenient). TAIL is POKEd to point to 1054. Then POKE in the scan code for the End key (which moves the cursor to the end of the line) and both the scan code and the ASCII value of Enter. When BASIC gets to the LINE INPUT, it finds the keys in the buffer, and executes them as if you typed End, Enter. This lets LINE INPUT read a line from the screen automatically. The process continues until all lines of the directory have been read.

The Temporary Array Is Erased

The directory lists two files per line, so the routine then splits each row into two entries in the array F\$. The temporary array TT\$, which was used to read each line, is then ERASEd, since it's no longer needed and would just take up memory.

You can use the subroutine starting at line 5000 in your own programs when you need to read and process the directory. The machinations are made invisible by printing the directory in the same color as the background. As per the comments, set FSPEC\$ to the filespec ("A:" or "B:" will do) and GOSUB 5000. You'll find all the filenames in the array F\$. The number of files, numbering from zero, can be found in the variable ENTRIES.

The only limitation is that the directory has to be printed on the screen in order to read it. A really long directory might scroll the screen, and the subroutine would be unable to read some of the names that have scrolled away. If this is a problem, just limit the number of files read by setting FSPEC\$. You could first read all BASIC files with FSPEC\$="A:\*.BAS", then .COM files, .BAT files, and so on.

### **Color Directory**

For error-free program entry, be sure to use "The Automatic Proofreader," Appendix B.

- A6 190 'Color Directory for The IBM Personal Computer and PCjr
- EC 110 'requires 40 columns, color board or PCjr with disk drive
- EE 115 'Standard BASIC
- # 12Ø SCREEN Ø,Ø,Ø:WIDTH 4Ø:COLOR 14,2,2:CLS:DEF INT A-Z:KEY OFF:FOR I=1 TO 1Ø:KEY I,"":NEX T
- ME 130 CR\$=CHR\$(17)+CHR\$(196)+CHR\$(217)
- 61 140 PRINT"Welcome to ";:COLOR 11:PRINT"Color D irectory"
- CI 145 PRINT: COLOR 1:PRINT"COMPUTE!'s SECOND BOOK of IBM"
- © 150 'Remove the word REM from following line for automatic use with drive A
- N 160 REM DRIVES="A:":FSPECS="A: \*. \* ": GOTO 200
- M 170 PRINT:COLOR 14:PRINT "Select Drive: (";:CO LOR 16,12:PRINT"A B";:COLOR 14,2:PRINT CHR \$(29);CHR\$(29);"/";CHR\$(28);")"
- P 18Ø DRIVE\$=INKEY\$+":":A=ASC(DRIVE\$):IF (A OR 3
  2)<97 OR (A OR 32)>98 THEN 18Ø
- JB 190 DRIVE\$=CHR\$(A AND 223)+":":FSPEC\$=DRIVE\$+"
  \*.\*"

```
MH 200 GOSUB 5000:COLOR 30,5,5:CLS:PRINT"Reading description file"
WP 210 DIM D$(ENTRIES):FOR I=0 TO ENTRIES:D$(I)=C
      HR$ (9) +"--" : NEXT
6P 22Ø DISKNAME$=" ":ON ERROR GOTO 31Ø
PA 230 OPEN DRIVE$+"DESCR.DIR" FOR INPUT AS #1
MG 240 LINE INPUT #1, DISKNAMES: LINE INPUT#1, AS: NU
      MREC=VAL (A$)
IN 250 FOR ITEMS=0 TO NUMREC
CC 260
       LINE INPUT #1.F$:LINE INPUT#1.D$
PO 27Ø
        FOR I=Ø TO ENTRIES
CF 28Ø
           IF F$=F$(I) THEN D$(I)=D$
#J 29Ø
         NEXT: NEXT
DI 300 GOTO 320
CP 310 RESUME 320
NJ 320 CLOSE#1:ON ERROR GOTO 0
IP 330 PAGES=INT (ENTRIES/8)
NL 340 CURR=0
PN 350 START=CURR$8:FINISH=START+7:IF FINISH>ENTR
      IES THEN FINISH=ENTRIES
HF 360 COLOR ,1,1:CLS:COLOR ,10:PRINT STRING$(80,
      32):LOCATE 1,2:COLOR 11:PRINT"Color Direct
      ory"; TAB(30); "Drive "; DRIVE$:LOCATE 2,20-L
      EN(DISKNAME$)/2:COLOR 15:PRINT DISKNAME$:C
      OLOR ,1:PRINT
6K 37Ø FOR I=START TO FINISH
HM 380 COLOR 0,6:PRINT "F";MID$(STR$(1+I-START),2);:COLOR 14,1:PRINT " ";F$(I);TAB(16);:COL
      OR 13:PRINT LEFT$(D$(I),25);:COLOR 3:PRINT
       STRING$ (40, 196);
08 39Ø NEXT
08 400 LOCATE 24,1:COLOR 9,7:PRINT"Press ";:COLOR
       Ø.6:PRINT"F1"::COLOR 9.7:PRINT" to "::COL
      OR Ø.6: PRINT "F"; MID$ (STR$ (1+FINISH-START)
       ,2);:COLOR 9,7:PRINT" to select program. "
       :TAB(40)::LOCATE 25.1:PRINT"Press PoUp or
      PgDn to page, ESC to quit.";
M 410 COLOR 12,1:LOCATE 22,12:PRINT"Page #";CURR
      +1; "of"; PAGES+1
CK 420 AS=INKEYS: IF AS="" THEN 420
6H 43Ø IF A$<>CHR$(27) THEN 54Ø
#0 440 COLOR 14,4:GOSUB 2000:PRINT"1. Exit to BAS
          2. Exit to DOS";:LOCATE 25,1:PRINT"3.
      Re-RUN 4. Save descriptions 5. Menu";
BF 450 A$=INKEY$: IF A$<"1" OR A$>"5" THEN 450
DB 460 ON VAL (A$) GOTO 470,480,490,500:GOTO 350
M 470 COLOR 7,0,0:CLS:END
IN 48Ø SYSTEM
```

F6 500 ON ERROR GOTO 510:GOSUB 1000:GOTO 350

HA 490 RUN

- IM 510 BEEP:COLOR 15,12:GOSUB 2000:COLOR 30:PRINT
   "Can't save descriptions. ";:LOCATE 25,1:
   COLOR 15:PRINT"Press ";CR\$;" to continue."
  :
- EE 520 IF INKEY\$<>CHR\$(13) THEN 520
- FN 53Ø RESUME 35Ø
- M 540 IF A\$=CHR\$(0)+CHR\$(81) THEN CURR=-(CURR+1)
  \*(CURR<PAGES):GOTO 350
- NH 55Ø IF A\$=CHR\$(Ø)+CHR\$(73) THEN CURR=CURR-1:CU RR=CURR-(PAGES+1)\*(CURR<Ø):GOTO 35Ø
- AK 56Ø A=ASC(MID\$(A\$+"Ø",2))-59:IF A<Ø OR A>FINIS
  H-START THEN BEEP:GOTO 42Ø
- © 570 COLOR 0,11:GOSUB 2000:PRINT"Press ";CR\$;"
  to run program, ESC to";:LOCATE 25,1:PRINT
  "return to menu, SPACE to do description";
- KO 580 LOCATE 4+A\*2,4:COLOR 26,1:PRINT F\$(START+A);:COLOR 0,11
- L 590 A\$=INKEY\$:IF A\$<>CHR\$(13) AND A\$<>CHR\$(27) AND A\$<>CHR\$(32) THEN 590
- 6K 6ØØ IF A\$=CHR\$(27) THEN COLOR 14,1:LOCATE 4+A\*
  2,4:PRINT F\$(START+A);:GOTO 4ØØ
- IN 610 IF A\$<>CHR\$ (32) THEN 670
- L! 620 IF DISKNAME\$=" " THEN GOSUB 2000:LOCATE 25,1:PRINT"Enter name of disk";:LOCATE 24,1:
  LINE INPUT;">",DISKNAME\$:IF DISKNAME\$="" THEN DISKNAME\$=" "
- 00 630 GOSUB 2000: Z=START+A:PRINT "Enter descript
  ion:";:LOCATE 25,1:PRINT D\$(Z);:LOCATE 25
  ,1:LINE INPUT;">";D\$(Z):D\$(Z)=LEFT\$(" "+D
  \$(Z),57):GOTO 350
- 60 640 COLOR 30,4:GOSUB 2000:BEEP:PRINT"Cannot sa ve descriptions to disk.";:COLOR 15:LOCATE 25,1:PRINT"Run program anyway? (Y/N):";
- MA 650 A\$=INKEY\$: IF A\$<>"y" AND A\$<>"Y" AND A\$<>"
  n" AND A\$<>"N" THEN 650
- LI 660 IF A\$="Y" OR A\$="Y" THEN RESUME 680 ELSE R ESUME 350
- EH 670 ON ERROR GOTO 640: GOSUB 1000
- JK 680 ON ERROR GOTO 690:COLOR 7,0,0:CLS:RUN DRIV E\$+F\$(START+A)
- JL 69Ø COLOR 3Ø,4:GOSUB 2ØØØ:BEEP:PRINT"Cannot ru
  n ";F\$(A);".";:COLOR 15:LOCATE 25,1:PRINT"
  Press ";CR\$;" to continue...";
- EA 700 IF INKEY\$<>CHR\$ (13) THEN 700
- FK 710 RESUME 350
- LE 72Ø END
- AD 1000 'Save descriptions to disk
- EK 1010 OPEN DRIVE\$+"DESCR.DIR" FOR OUTPUT AS #1
- MB 1020 PRINT#1, DISKNAME\$; CHR\$(13); ENTRIES; CHR\$(1
  3);

- W 1030 FOR I=0 TO ENTRIES:PRINT#1,F\$(I);CHR\$(13)
  ;D\$(I);CHR\$(13);:NEXT
- CH 1040 CLOSE #1: ON ERROR GOTO 0: RETURN
- IK 1Ø5Ø '
- N 2000 LOCATE 24,1:PRINT SPACE\$(39);:LOCATE 25,1 :PRINT SPACE\$(39);:LOCATE 24,1:RETURN
- AB 5000 'This subroutine reads disk directory int o a string array
- PD 5010 'Enter with FSPEC\$, the file spec for the FILES command
- El 5020 'Exits with array F\$, and NUMFILES, the number of files
- HN 5030 'uses a temporary array, TT\$, which is ER ASEd after use
- IL 5949 '
- KB 5050 DEF SEG=0
- JM 5060 HEAD=1050: TAIL=1052: BUFFER=1054
- KB 5070 CLS:COLOR 30:PRINT"Reading disk directory
- IC 5080 COLOR 2: ON ERROR GOTO 5100
- BK 5090 FILES FSPEC\$: ON ERROR GOTO 0:GOTO 5110
- © 5100 BEEP:COLOR 31:CLS:PRINT"Cannot read directory":COLOR 7:ON ERROR GOTO 0:END
- IM 5110 DIM TT\$(24):LOCATE 3,1:COLOR 7:ROWS=0
- MK 5120 'Put code for End, Enter into keyboard bu ffer:
- EH 5130 POKE HEAD, 30: POKE TAIL, 34: POKE BUFFER, 0: P OKE BUFFER+1, 79: POKE BUFFER+2, 13: POKE BUF FER+3, 28
- FK 5140 LINE INPUT TT\$ (ROWS)
- EL 5150 IF TT\$(ROWS)<>"" THEN ROWS=ROWS+1:GOTO 51
- PA 5160 IF NOT DIMMED THEN DIM F\$(ROWS\*2-1):DIMME D=1
- KI 5170 ROWS=ROWS-1
- KP 5180 FOR I=0 TO ROWS
- JO 5190 FOR J=0 TO 1
- FF 5299 T\$=MID\$(TT\$(I),J\$18+1,12)
- PJ 5210 IF T\$<>"" THEN F\$(ENTRIES)=T\$:ENTRIES= ENTRIES+1
- NB 522Ø NEXT J
- 6A 523Ø NEXT I
- ON 5240 ERASE TT\$: ENTRIES=ENTRIES-1
- 01 5250 DEF SEG:RETURN

### **Memo Diary**

Jim Butterfield

Keep track of important dates, holidays, and personal events with this simple, easy-to-use BASIC program. For the IBM PC and Enhanced Model PCjr (disk only).

"Memo Diary" helps you record and recall birthdays, holidays, appointments, or any other event worth remembering. The program maintains a data file of up to 100 events, whose dates can range from tomorrow to one year in the future. You can record two different types of dates—temporary, one-time events such as appointments which have no importance once they've passed, and permanent, recurring events such as birthdays and anniversaries. By routinely running Memo Diary each time you use your computer, you won't have to worry about forgetting to mail a birthday card to a relative or finding an anniversary gift for a spouse.

The program always shows the correct day of the week when you enter a date, and you need to enter the year only once—the very first time you run the program. After that (for the next 99 years, anyway) Memo Diary keeps track of the year for you. Each time you run the program, it automatically shows all due and overdue events on the screen or printer, and erases one-time events from the calendar after they're

displayed.

You can enter temporary or recurring new events and erase existing events whenever you wish. You can also examine all events from the current date forward, or search the entire calendar for events matching a given starting pattern. Finally, Memo Diary saves your calendar to disk.

### The First Time

The first time you run Memo Diary is special. Do not start the program by entering RUN. Type RUN 100 and press ENTER. If you don't do this, the program will not work correctly. When you start the program at line 100, Memo Diary lets you enter the correct year without looking for a previous file of events. Thereafter, start the program with RUN in the usual way.

On the first run you'll probably want to enter fixed holidays such as New Year's Day as well as birthdays and anniversaries. These are permanent events which you won't need to enter year after year. Holidays which fall on a different date each year, like Thanksgiving or Easter, should be entered as one-time events.

When Memo Diary asks you to enter today's date, you can type in the name of the month (such as OCTOBER) or its number (such as 10). In either case, be careful to enter it correctly. Memo Diary lets you enter any day of the month from 1 to 31, so it won't mind if you specify the date as February 30. Mistakes like these may confuse the calendar file. For instance, if you use the program on July 4, and the next day mistakenly give the date as June 5, the computer thinks you've let almost a whole year go by. To warn you of this, Memo Diary displays HAPPY NEW YEAR. If you see this message when a new year hasn't arrived, stop the program and start over, entering the correct date.

A Memory Jogger

Except for the very first run, Memo Diary always begins by reporting all due and overdue events (You just missed your anniversary). Take careful note of these events, since they'll soon be erased from the calendar (if they're temporary events) or moved ahead to next year (if they're permanent). To help jog your memory, Memo Diary also lets you make a copy of the list of events on your printer.

After disposing of due and overdue events, Memo Diary displays five options: You can see future events, add a new event, cancel an event, search for an event, or quit the program. You'll ordinarily want to look ahead to see what's coming in the next week or two. To do this, choose option 1 (see future events) and supply an appropriate future date when requested. If you enter the current date when looking at future events, Memo Diary assumes you mean the same date *next* year and gives you everything on file.

When you want to make a new entry, select option 2 (add new event). Memo Diary first asks whether the new event is one-time or permanent. It then lets you enter the date and details. Again, the current date is understood as one year from today (it's assumed you don't need to record an event that's

happening the same day).

To cancel an event (option 3), you must know its date. When an event is entered, you're shown every item scheduled for that date, each with its own code number. To cancel an event, type in its code number when prompted.

Option 4 (search for event) lets you search for an event based on the first few letters of the entry. You may find many events in such a search. For instance, if the calendar file contains the events CLUB MEETING, CLUB CONFERENCE, and CLUB ELECTION, searching for CLUB displays all three events. In this case you would *not* see the entry CANADIAN CLUB, since CLUB is spotted only if it's in the first word of the entry. Thus, if you plan to search for certain keywords (BIRTHDAY, CHURCH, SOFTBALL, or whatever), keep them at the front of each calendar entry.

After you've finished an option, Memo Diary always returns you to the main menu. Sooner or later you'll be ready to use option 5 (quit). The program knows when it's time to update the calendar file. If you've erased past and overdue events, added or deleted items, Memo Diary will—with your permission—proceed to update the data file on disk or tape.

### The Time Pivot

A program that handles dates can encounter some subtle paradoxes. Does August come before April, or after it? The correct answer is *both*. Memo Diary could resolve this difficulty by adding a year designation to every event, but that complicates the handling of permanent events which don't belong to a specific year. This is not a trivial problem: If you schedule a new event for August, the program must decide whether to add the event to the calendar ahead of an existing April event or after it. Without a year designation, how can anyone tell?

The problem is solved by using a *pivot* date, usually the same as the current date. If today is July 4, August does indeed come *before* April. On the other hand, if today is November 11, *April* comes before August. Since the calendar always looks one year into the future, everything is kept in order.

However, there's one case in which the pivot date can't be the current date. Each time the program begins, it must measure the time lapse since its last use. For example, say that you last used the program on August 20, 1986, and next use it on September 4, 1986. On the first run (August 20), Memo Diary uses August 20 as the pivot. That way an event dated Sep-

tember 1 is seen ahead of another item dated in October.

On the second run (September 4) the September 1 event is reported as past due and either erased from the calendar (if it's temporary) or moved ahead to September 1 of next year (if it's permanent). Once this is done, the pivot date moves forward to September 4, meaning that a September 1 event now belongs *after* an item dated in October. Don't worry if this sounds confusing—it works out more simply in practice than in theory.

The day of the week is calculated with a simple formula. If you haven't seen it before, here's a hint on how it works. The calendar is modified to make March 1 the first day of the "adjusted year." This way, leap year with its extra February 29 date doesn't break up the sequence of days—the extra leap day just gets pasted onto the year's end. Though the math is a bit convoluted, you may find it interesting to trace the logic of this routine (it starts at line 2150 in the listing).

### Expanding the Calendar

Memo Diary can keep track of a maximum of 100 events. In practice it's wise to limit the number to 80 or 90 to leave room for permanent events that move automatically from the front to the back of the list. If you need more than 100 events, change the L\$ value in the DIM statement in line 150. You can increase the 100 to whatever number you like, but don't get carried away. Since Memo Diary uses string arrays, a very large value may cause garbage collection delays. There's no particular limit to the number of events allowed for a particular date.

### **Program Notes**

Let's take a look at the program's major features. Line 90 prepares Memo Diary to read a file. The variable F is a *Boolean* (logical) variable that's defined as *true* here, to let you read the calendar file on a normal run. When you enter at line 100 on the first run, F is *false* (like every other undefined variable), and no file is read.

DATA statements in lines 110–140 hold the names of the months of the year and days of the week; the names are read into the arrays M\$ and W\$. Line 150 dimensions the L\$ array for 100 items. Lines 230–250 call for a reading of the calendar

file if appropriate. This is done in the subroutine at line 3010. When Memo Diary reads this file, it detects and reports the last date the file was used. Line 260 asks for today's date; the subroutine at line 1670 asks for and accepts the date.

Now it's time to search for due and overdue events. Using the previous date as a pivot, the subroutine at line 1960 scans for all events up to today's date. The program reports these events, erases them, or moves them ahead as needed, and proceeds to the main menu. Line 680 begins a main activity loop—it prompts with the menu, asks for a choice, then goes to the appropriate subroutine. Line 850 lets you see future events. Since the pivot date is now today, the program scans to the requested future date to see how many events fall into the today-to-future-date range.

Line 940 lets you add a new event. After asking AN-NUAL OR ONE-TIME?, the program requests the event's date and then asks for details. After adding a year designation to the date of one-time events, the new event is inserted into the proper sequence. Line 1210 lets you cancel an event. Memo Diary asks for a date and then lists all events that match that date. At line 1350, the program asks which event to delete. Note that the number you supply must be in the correct range.

Line 1450 begins the search-for-an-event routine. After it receives a search string (P\$), the program looks for a match. When it scans through the calendar, it must look in different places depending on whether the event is one-time or permanent. That's because one-time events carry a year designation, making their dates three characters longer.

### A Horrible Mistake?

Line 1570 handles the quit option; the flag F9 registers activity. If you haven't changed any of the data, there's no need to update the calendar file. Before scratching the old file and writing the new one, the program asks whether you're ready. That way, if you made some horrible mistake, you can cancel the file update.

The main loop ends at line 1580 and is followed by several subroutines. The routine starting at line 1590 writes a new calendar file when appropriate, and line 1670 begins the date input routine. The date is formed into a string (D8\$) to allow for easy searches or entry. The subroutine at line 1930 reads

the calendar file. The first item in the file is always the most recent date of use; the remaining data is events.

The subroutine at line 1960 scans all events to see which have dates between the pivot date (D9\$) and a second date (D8\$). There are three dates involved: event, pivot, and the second date, which makes the comparison a bit messy. Boolean variables keep everything in order. Eventually, the variable F0 indicates the date is in range, and the variable L0 indicates when the last event is found within the date range.

The routine starting at line 4020 displays the information, on the printer if desired. The date is given complete with the day of the week, and events falling on the same day are grouped together. The weekday calculation begins at line 2150. The weekday variable, W, ranges from 0 to 6, so 0 means Sunday. As written, this routine is good for years ranging from 85 (1985) to 84 (2084). If you want to plan more than 99 years in advance, you'll need to modify the routine.

### Memo Diary

For error-free program entry, be sure to use "The Automatic Proofreader," Appendix B.

```
ME 9Ø F=(1=1)
DM 100 GOSUB 2250
KL 105 WIDTH 80:KEY OFF: DEF SEG-0:POKE 1047, PEEK (
      1047) OR 64
JO 110 DATA JAN, FEB, MAR, APR, MAY, JUN
FD 120 DATA JUL, AUG, SEP, OCT, NOV, DEC
MP 130 DATA SUNDAY, MONDAY, TUESDAY, WEDNESDAY
JJ 140 DATA THURSDAY, FRIDAY, SATURDAY
IN 150 DIM M$(12), W$(6), L$(100)
JC 160 FOR J=1 TO 12
6M 17Ø READ M$(J)
88 18Ø NEXT J
JL 190 FOR J=0 TO 6
LD 200 READ W$(J)
NE 210 NEXT J
BN 220 FRINT "EVENT CALENDAR"
NM 23Ø IF F=Ø THEN 26Ø
10 24Ø C=1
PS 250 GOSUB 3010
PM 260 PRINT "TODAY'S DATE:"
08 27Ø Y8=Y9
IM 28Ø GOSUB 167Ø
KI 290 M8=M
BC 300 D8=D
#K 310 IF MB)=M9 THEN 330
PB 32Ø Y8=Y9+1
```

```
DG 33Ø IF M8<>M9 OR D8>=D9 THEN 35Ø
PF 34Ø Y8=Y9+1
KC 35Ø IF Y8<=Y9 THEN 37Ø
CP 360 PRINT "HAPPY NEW YEAR"
IA 370 IF F THEN 400
6M 38Ø PRINT "YEAR":
# 390 INPUT Y8
LC 400 D9$=RIGHT$(STR$(100+M9),2)+"/"
0J 41Ø D9$=D9$+RIGHT$(STR$(1ØØ+D9),2)
H 420 IF F THEN 440
61 43Ø D9$=D8$
PK 44Ø F= (1=1)
JF 45Ø GOSUB 196Ø
M 460 PRINT "PAST EVENTS: ":
NL 47Ø IF LØ>=Ø THEN 5ØØ
SD 480 PRINT "NONE"
IK 49Ø GOTO 65Ø
CF 500 PRINT L0+1
M 510 GOSUB 4010
N 52Ø F9=-1
BD 53Ø FOR J=Ø TO LØ
PD 54Ø IF MID$(L$(J),6,1)="/" THEN 57Ø
09 55Ø L$(L9)=L$(J)
61 56Ø L9=L9+1
00 57Ø NEXT J
AF 58Ø L8=LØ+1
CF 59Ø FOR J=L8 TO L9-1
PM 600 L$(J-LB)=L$(J)
DI 610 NEXT J
00 62Ø L9=L9-L8
KD 630 L8=0
HE 640 L=L9
NN 650 F=0
JB 660 F9=0
6C 67Ø D9$=D8$
#8 68Ø L=L9-L8
KB 69Ø IF L<>Ø THEN 71Ø
II 700 PRINT "NO FUTURE EVENTS"
CM 71Ø IF L=Ø THEN 73Ø
CN 720 PRINT L; " FUTURE EVENTS"
JN 730 PRINT
E8 740 PRINT "1. SEE FUTURE EVENTS"
CI 750 PRINT "2. ADD NEW EVENT"
NO 760 PRINT "3. CANCEL EVENT"
KE 770 PRINT "4. SEARCH FOR EVENT"
№ 78Ø PRINT "5. QUIT"
KI 79Ø PRINT
KS 800 PRINT "...YOUR CHOICE (1-5)";
NC 810 INPUT A
JL 82Ø PRINT
```

```
KA 830 ON A GOTO 850,940,1210,1450,1570
HI 840 GOTO 730
IB 850 PRINT "AHEAD TO DATE:"
10 86Ø GOSUB 167Ø
JH 87Ø GOSUB 196Ø
JK 88Ø IF LØ<>-1 THEN 91Ø
MI 890 PRINT "NO EVENTS"
HK 900 GOTO 920
AA 910 GOSUB 4010
FI 920 PRINT L9-L0-1; " OTHER FUTURE EVENTS"
6H 93Ø GOTO 73Ø
CI 940 PRINT "ANNUAL OR DNE-TIME (A/O)":
CA 950 INPUT P$
MI 960 A=0
AG 970 P$=LEFT$ (P$, 1)
KL 98Ø IF P$="0" THEN 1010
MF 990 A=1
6F 1000 IF P$<>"A" THEN 730
MM 1010 GOSUB 1670
0! 1020 Y$="/"+RIGHT$(STR$(101+Y8),2)
SA 1030 IF D8$<=D9$ THEN 1050
MD 1040 Y$="/"+RIGHT$(STR$(100+YB),2)
QL 1050 IF A<>1 THEN 1070
BD 1060 Y$=""
OH 1070 GOSUB 1960
DN 1080 IF L9-1<L0+1 THEN 1120
6A 1090 FOR J=L9-1 TO L0+1 STEP -1
IP 1100 L$(J+1)=L$(J)
SP 1110 NEXT J
HG 1120 PRINT "DETAIL":
N 1130 INPUT LLS
ON 1140 D8$=D8$+Y$
MC 115Ø D8$=D8$+" "
CD 1160 L$(LØ+1)=D8$+LL$
OC 117Ø L9=L9+1
HP 1180 L=L9
E! 119Ø F9=-1
SB 1200 GOTO 580
HA 121Ø PRINT "CHANGE WHICH DATE:"
MD 1220 GOSUB 1670
BN 1230 LØ=-1
NR 1240 FOR J=L8 TO L9-1
KO 1250 IF D8$<>LEFT$(L$(J),5) THEN 1300
E! 1250 L1=J
DF 127Ø IF LØ<>-1 THEN 129Ø
DH 128Ø LØ=J
FE 1290 PRINT J;": ";L$(J)
GA 1300 NEXT J
NI 131Ø IF LØ<>-1 THEN 134Ø
BO 1320 PRINT "NO EVENTS"
```

```
EL 133Ø GOTO 73Ø
8 1340 PRINT
IH 1350 PRINT " DELETE WHICH EVENT ABOVE";
KC 1360 INPUT A
NN 137Ø IF AKLØ OR ADL1 THEN 73Ø
MN 1380 FOR J=A TO L9-1
D6 139Ø L$(J)=L$(J+1)
60 1400 NEXT J
PM 1410 L9=L9-1
DJ 142Ø F9=-1
DA 1430 PRINT " ... DELETED"
HB 144Ø GOTO 68Ø
6A 145Ø PRINT "SEARCH FOR";
DM 1460 INPUT P$
JH 1470 P=LEN(P$)
PD 148Ø FOR J=Ø TO L9-1
JN 149Ø A=7
6J 1500 IF MID$(L$(J),6,1)<>"/" THEN 1520
PH 1510 A=10
EA 1520 IF A+P-1>LEN(L$(J)) OR P$<>MID$(L$(J),A,P
       ) THEN 1540
OK 153Ø PRINT L$(J)
HA 154Ø NEXT J
CL 155Ø PRINT "
                    END OF SEARCH"
FI 156Ø GOTO 73Ø
0M 157Ø IF F9<>Ø THEN 159Ø
ID 158Ø END
EC 1590 PRINT "READY TO WRITE NEW EVENT FILE (Y/N
       ) ";
00 1600 INPUT P$
PK 1610 IF LEFT$ (P$, 1) = "Y" THEN 1630
CJ 1620 STOP
NJ 163Ø D9$=D9$+"/"
BJ 164Ø D9$=D9$+RIGHT$(STR$(Y8+1ØØ),2)
HJ 165Ø €=2
KH 166Ø GOTO 3Ø1Ø
KL 167Ø M=Ø
KD 168Ø PRINT "MONTH";
CC 169Ø INPUT MM$
JF 1700 M=VAL (MM$)
PE 1710 MM$=LEFT$ (MM$+"XX",3)
NK 172Ø IF M=Ø THEN 176Ø
LM 1730 IF M<1 DR M>12 THEN 1670
CK 174Ø PRINT M$(M)
PA 175Ø GOTO 181Ø
JL 176Ø FOR J=1 TO 12
IF 177Ø IF MM$<>M$(J) THEN 179Ø
HA 178Ø M=J
10 1790 NEXT J
KF 1800 IF M<1 OR M>12 THEN 1670
```

```
EN 1810 PRINT "DAY":
HE 1820 INPUT D
00 183Ø IF D<1 OR D>31 THEN 167Ø
NA 1840 D8$=RIGHT$(STR$(100+M),2)+"/"
NM 185Ø D8$=D8$+RIGHT$(STR$(100+D),2)
CF 1860 Y=Y8
#1 187Ø IF D8$>=D9$ THEN 189Ø
PB 188Ø Y=Y8+1
JB 1890 GOSUB 2150
HN 1900 IF LEN(LL$) <=0 THEN 1920
EC 191Ø PRINT "(":W$(W):")"
JD 1920 RETURN
68 193Ø C=1
FN 1940 GOSUB 3010
KM 195Ø RETURN
BN 1960 LL$=CHR$ (255)
CH 1970 LØ=-1
PN 1980 IF L<>0 THEN 2000
LI 199Ø RETURN
10 2000 V$=D8$+LL$
AA 2010 WW$=D9$
PI 2020 IF F<>0 THEN 2040
88 2030 WW$=D9$+LL$
DL 2040 F1=(WW$>=V$)
NB 2050 FOR J=LB TO L9-1
LG 2060 F2=(L$(J)>WW$)
GH 2070 F3=(V$)L$(J))
KN 2080 F0=F2 AND F3
6J 2090 IF F1=0 THEN 2110
80 2100 FØ=F2 OR F3
60 2110 IF FØ=Ø THEN 2130
CE 2120 LØ=J
66 213Ø NEXT J
JK 2140 RETURN
EE 2150 IF Y>=85 THEN 2170
0M 21AØ Y=Y+1ØØ
CE 217Ø M1=M+1
M 218Ø M2=INT(1/M1+.7)
EE 219Ø M3=Y-M2
DF 2200 M4=M1+12*M2
FB 221Ø N=INT(M4*3Ø.6ØØ1)+INT(M3*365.25)+D
CG 222Ø M6=INT(N/7)
JH 223Ø W=N-7*M6
JM 224Ø RETURN
ND 225Ø CLS
JC 226Ø RETURN
NE 3000 REM INPUT/OUTPUT ROUTINE
FD 3010 ON ERROR GOTO 3100
MA 3020 F$="EVENTS": INPUT "ENTER DRIVE # (IE., A)
        : ":FF$:F$=FF$+":"+F$
```

- EE 3030 IF C=2 THEN 3080
- PM 3040 OPEN F\$ FOR INPUT AS #1:INPUT #1,LL\$:IF L EN(LL\$)<>8 THEN PRINT LL\$:"?":60T0 3070
- FN 3060 INPUT #1,L\$(L):L=L+1:IF EOF(1)=0 THEN 306
- PA 3070 CLOSE #1:ON ERROR GOTO 0:L8=0:L9=L:RETURN
- C6 3080 OPEN F\$ FOR OUTPUT AS #1:PRINT #1,D9\$:FOR J≃0 TO L9-1:PRINT #1,L\$(J)
- NB 3090 NEXT J:CLOSE #1:ON ERROR GOTO 0:END
- EC 3100 CLOSE #1:PRINT "DISK ERROR #";ERR; "OCCURR ED.":PRINT "TRY AGAIN."
- HH 3110 PRINT: PRINT "HIT A KEY TO CONTINUE"
- FK 3120 A\$=INKEY\$:IF A\$="" THEN 3120
- JA 313Ø RESUME 3020
- M 4000 REM PRINT ROUTINE
- 83 4010 ON ERROR GOTO 4090
- BA 4020 D\$="":INPUT "WANT EVENTS ON PRINTER (Y/N)
  ":P\$
- P0 4030 IF LEFT\$(P\$,1)="Y" THEN OPEN "LPT1:" FOR OUTPUT AS #1 ELSE OPEN "SCRN:" FOR OUTPUT AS #1
- CH 4040 FOR J=L8 TO L0:IF D\$=LEFT\$(L\$(J),5) THEN 4080
- P 4050 D\$=LEFT\$(L\$(J),5):M=VAL(LEFT\$(D\$,2)):D=VA
  L(MID\$(D\$,4,2))
- NJ 4060 Y=Y8: IF D\$<D9\$ THEN Y=Y8+1
- 6L 4070 GOSUB 2150:PRINT #1,W\$(W);" ";:PRINT #1,M \$(M);D
- PL 4080 PRINT #1," ";MID\$(L\$(J),4):NEXT J:CLOSE #1:ON ERROR GOTO 0:RETURN
- IC 4090 CLOSE #1:PRINT "PRINTER ERROR #";ERR;"OCC URRED.":PRINT "TRY AGAIN."
- HF 4100 PRINT: PRINT "HIT A KEY TO CONTINUE"
- EG 4110 AS=INKEYS: IF AS="" THEN 4110
- KL 4120 RESUME 4020

### **Fast Filer**

### Richard Mansfield and Patrick Parrish

Maintain a master index of magazine articles with this short BASIC program for the IBM PC and PCjr.

How many times have you been working on a program when you recall a magazine article that has just the information you need—but finding it is another matter? That is, you know the article's *somewhere* in the house—but where? You could spend hours paging through back issues to find what you're looking for. Now, with "Fast Filer," you'll have a fast and easy way to retrieve such information.

Enter and save Fast Filer. Be sure to use "The Automatic Proofreader," found in Appendix B, when you type in the program.

### Searching the Database

Fast Filer is simple and convenient. To search the database, all you really need to do is type RUN and follow the prompts. The program first asks whether you want to send output to the screen or the printer. Then the menu displays several options. You can search the database in several different ways—by magazine title, by author, by subject, by publication date, or by two categories at once.

For example, say you want to list all articles from COM-PUTE! magazine. Simply choose option 1 and enter COM-PUTE! when prompted for the magazine name. To list all articles by, say, Charles Brannon, choose option 2 and enter BRANNON in response to the author prompt. Once the listing begins, you can pause it by pressing any key and resume by pressing P.

Fast Filer accepts abbreviations, so it's usually not necessary to type in the entire name. You can abbreviate COM-PUTE! as COMPU, for instance. However, you must give Fast Filer enough information to distinguish similar names. If the database contains articles by Butterfield and Buncombe, entering BU for the author lists all articles by *both* authors, since both names share those two characters. Entering BUT would distinguish the two names and list all Butterfield articles.

For added flexibility, options 5 and 6 let you search by more than one category at a time. Option 5 provides an AND function to find articles that *share* two categories. To find all *COMPUTE!* articles written by Charles Brannon, for example, select option 5 and enter 1,2 (be sure to separate the numbers with a comma). Then enter the magazine and author names as prompted.

Option 6 provides an OR function to find articles in *either* of two categories. Perhaps you're interested in machine language. With option 6 you could find every article that was categorized under the subject MACHINE LANGUAGE, or that was written by Jim Butterfield (who often writes on that subject). The ability to search two categories simultaneously is

very powerful.

**Easy Data Entry** 

Of course, no database is useful until it contains some data. Line 1999 of Fast Filer is a template which shows the format for entering data. To enter new data, simply add new lines to Fast Filer, using line numbers higher than 1999. (Lines 2000–2040 are examples which you can modify or delete.)

Every new entry must be in the form of a BASIC line consisting of a line number and a DATA statement, followed by six data items separated by commas. Here's the format:

### MAGAZINE TITLE, AUTHOR, SUBJECT, DATE, PAGE NUMBER, COMMENTS

Because Fast Filer separates data items with commas, you must *not* put commas within the data itself. For instance, enter BRANNON C for an author's name, *not* BRANNON, C.

You cannot omit any of the data items for a particular entry. If you do, the entire list of data is thrown out of sequence. Instead of leaving a particular item blank, substitute something like N/A (for Not Applicable). For example, you might have an entry for which you don't need to add a comment, like:

### 2000 COMPUTE!, GASPARD, UTILITY, 9/84, 114, N/A

Pay particular attention to line 2050, which tells Fast Filer it's reached the end of the data. This must always be the last DATA line in the program. When adding new data, renumber this line accordingly. When you're finished adding data, resave Fast Filer on disk. The next time you run it, all the new data is

available. Since the data is appended to the program itself, the size of the database is limited only by your computer's memory.

Designing a Database

Fast Filer provides the basic framework for a database, but for maximum flexibility, it leaves the most important design choices up to you. You're free to choose whatever subject categories you like, making them as general or as specific as your needs require.

Creating categories deserves some careful forethought. Clearly, a subject category like COMPUTERS is too broad to be useful. On the other hand, the subject must have enough breadth to encompass more than one article. Consistency is essential, too. If you pick MACHINE LANGUAGE as a subject, then stick with that subject name—categorizing other machine language articles under subject names like ML or MACH LANG will result in incomplete searches.

Before adding your first entry, you may want to decide on standard names for your major categories. These could be saved for future reference in a written list or added to Fast Filer as REM statements.

Use consistent names for magazine titles and authors as well. If you enter a magazine title as COMPUTE (without the exclamation point), it won't be found when you search for articles under the keyword COMPUTE! (although the reverse would work). Likewise, GAZETTE is a more convenient title than COMPUTE!'S GAZETTE.

Fast Filer's ability to abbreviate can work to your advantage. For instance, say that you pick GRAPHICS as a major category. If you enter graphics articles under subject names like GRAPHICS PC and GRAPHICS PCJR, then Fast Filer can find *all* graphics articles (under the subject GRAPHICS) as well as graphics articles for a particular computer.

There are limits to what Fast Filer can do, of course, as there are with any BASIC program this short. But its simplicity makes the program easier to customize. One of the best ways to learn programming is to begin with an existing program and alter it to fit your own needs. Such changes can range from the purely cosmetic to more significant improvements (formatting printer output, adding extra categories, and so on). In fact, with only a few modifications, Fast Filer can be used

to index practically anything, from books or record albums to investments, rare coins, or stamps.

### **Fast Filer**

For error-free program entry, be sure to use "The Automatic Proofreader," Appendix B.

- 10 10 WIDTH 40:KEY OFF:DEF SEG=0:POKE 1047, PEEK(1 047) OR 64:DIM A\$(6):G\$=" "
- NL 20 CLS:LOCATE 10,3,0:PRINT "PRINT TO SCREEN OR PRINTER (S/P)?"
- KB 3Ø K\$=INKEY\$:IF K\$="" OR (K\$<>"P" AND K\$<>"S")
  THEN 3Ø
- 61 40 DE=-(K\$="S"): IF DE=1 THEN OPEN "SCRN:" FOR OUTPUT AS #1 ELSE OPEN "LPT1:" FOR OUTPUT A S #1
- CM 50 LABEL\$(1)="MAGAZINE TITLE:":LABEL\$(2)="AUTH OR'S LAST NAME:"
- N 60 LABEL\$(3)="THE TARGET SUBJECT:":LABEL\$(4)="DATE (IE., 1/14/85 OR 1/85):"
- % 70 CLS:PRINT STRING\$(6,31)G\$"CHOOSE ONE (1-8):
   ":PRINT:PRINT G\$" 1. MAGAZINE"
- JP 80 PRINT G\$" 2. AUTHOR":PRINT G\$" 3. SUBJECT":
   PRINT G\$" 4. DATE"
- PB 90 PRINT G\$" 5. AND":PRINT G\$" 6. OR":PRINT G\$
  " 7. PRINT ALL":PRINT G\$" 8. QUIT"
- LP 100 K\$=INKEY\$:IF K\$="" OR (VAL(K\$)<1 OR VAL(K\$)>8) THEN 100
- PE 110 K=VAL(K\$):ON K GOTO 120,130,140,150,160,16
- NF 120 C=1:GOTO 330
- 00 13Ø C=2:GOTO 33Ø
- OH 140 C=3:GOTO 330
- PA 150 C=4:GOTO 330
- FN 160 H\$="OR": IF K=5 THEN H\$="AND"
- KB 17Ø PRINT:PRINT G\$"# "H\$" # (1-4):":PRINT G\$;: INPUT N1,N2
- 相 180 IF (N1<1 DR N1>4) DR (N2<1 DR N2>4) THEN 1 70
- ## 190 CLS:PRINT "TYPE "LABEL\$(N1):INPUT I1\$:L=LE
  N(I1\$)
- EE 200 PRINT:PRINT "TYPE "LABEL\$(N2):INPUT I2\$:L2 =LEN(I2\$)
- PP 210 PRINT: Q=0:F=0:RESTORE
- FD 220 GOSUB 420: IF F=1 THEN 390
- 00 23Ø IF K=6 THEN 25Ø
- PI 240 IF LEFT\$(A\$(N1),L)<>I1\$ OR LEFT\$(A\$(N2),L2)
  )<>I2\$ THEN 270 ELSE 260
- PL 25Ø IF LEFT\$(A\$(N1),L)<>I1\$ AND LEFT\$(A\$(N2),L 2)<>I2\$ THEN 27Ø

- 3 260 Q=1:GOSUB 440
- JM 27Ø IF F=Ø THEN 22Ø
- IE 28Ø GOTO 39Ø
- LF 290 CLS:F=0:RESTORE
- HP 300 GOSUB 420: IF F=1 THEN 400
- JC 310 GOSUB 440: IF F=0 THEN 300 ELSE 400
- ID 320 CLOSE #1: END
- AR 330 CLS:PRINT "TYPE "LABEL\$(C):INPUT IN\$:L=LEN (IN\$)
- PS 34Ø PRINT: Q=Ø:F=Ø:RESTORE
- 5% 35Ø GOSUB 42Ø: IF F=1 THEN 39Ø
- EI 36Ø IF LEFT\$(A\$(C),L)<>IN\$ THEN 38Ø
- JD 370 Q=1:60SUB 440
- NG 380 IF F=0 THEN 350
- EL 390 IF Q=0 THEN PRINT:PRINT G\$;:COLOR 0,7:PRIN T"NO MATCHES FOUND":COLOR 7,0:PRINT
- IH 400 PRINT " "G\$;:COLOR 0,7:PRINT "PRESS ANY KE
  Y":COLOR 7,0
- 00 410 A\$=INKEY\$:IF A\$="" THEN 410 ELSE 70
- AG 420 READ A\$(1),A\$(2),A\$(3),A\$(4),A\$(5),A\$(6):I F A\$(1)="END" THEN F=1
- ME 43Ø RETURN
- 0M 44Ø PRINT #1,A\$(1)" "A\$(2)" "A\$(3)" "A\$(4)" P. "A\$(5)" "A\$(6):PRINT #1,
- LK 450 AS=INKEYS: IF AS="" THEN RETURN
- 00 460 A\$=INKEY\$: IF A\$="" OR A\$<>"P" THEN 460
- NM 47Ø RETURN
- HK 1999 REM DATA TEMPLATE: MAGAZINE, AUTHOR, SUBJEC T. DATE, PAGE, COMMENTS
- JI 2000 DATA COMPUTE!, SCHULTZ N, MINDBUSTERS, 4/85, 44, GAME
- LH 2010 DATA GAZETTE, BRANNON C, HORIZONS, 1/85, 80, V IC TO 64
- BH 2020 DATA GAZETTE, RANDALL N, ROAD TO MOSCOW, 12/ 84,80,GAME REVIEW
- DM 2030 DATA COMPUTE!, WATSON D, APPLE SCREEN DUMP, 10/84,169, TEXT SCREEN
- M 2040 DATA COMPUTE!, KEES M, SUPERBASIC 64,10/83, 198, ADDS 35 COMMANDS TO BASIC
- # 2050 DATA END, 0, 0, 0, 0, 0

### Screen Clock

### Marc Sugiyama

Have you ever become submerged in a project while working on your computer and suddenly discovered it is hours past your bedtime? Or maybe you need to keep a detailed log of your worktime on the computer for business or tax purposes. If so, "Screen Clock" is the answer—it constantly displays all this information and more on your monitor screen. Works with IBM PC and PCjr computers using DOS 2.0 or higher.

Large mainframe computers generally provide a *sysline* on the terminal screen which tells you the current date and time, who has logged on or off, and whether you've received any new electronic mail. Obviously, not all of these things apply to single-user personal computers, but some of the features would be nice to have.

"Screen Clock" is a short machine language program that prints the day of the week, date, current time, and log-on time at the top of the screen. This information appears no matter what else your computer is doing. You can be running a word processor, copying files, programming, or whatever—the day, date, and time will always be visible.

You might be wondering how it's possible to keep Screen Clock active while running another program—an IBM PC with PC-DOS, after all, isn't capable of *multitasking*, the ability to run two or more programs simultaneously. Screen Clock gets around this restriction by not using any PC-DOS function calls, relying instead on the BIOS (Basic Input/Output System) to handle the screen. This has several fortunate consequences:

- Sysline updates are not redirected to a file if you're using DOS file redirection.
- Updates are not printed if you're echoing output to the printer. (But the sysline *is* printed if you press PrtSc for a screen dump.)
- Screen Clock always updates the current "active" screen. It doesn't matter if you switch from the monochrome monitor to the color monitor, change pages in the color screens, or

even enter a graphics mode—the date and time are always there.

### Winding Up the Clock

Type in the program, save a copy on disk, then enter RUN. The program is a BASIC loader that creates a machine language file on your disk with the filename CLOCK.COM. To start the clock, simply type *CLOCK* (uppercase or lowercase is fine) at the A> DOS prompt. A sysline similar to this should appear on the top line of your screen:

### Wed Jan 01, 1986 12:01A (00:37)

The day of the week, date, and current time are self-explanatory. The figure in parentheses is the elapsed time (in hours and minutes) since Screen Clock was started or reset. This log-on time runs up to 23 hours and 59 minutes, then rolls over to 00:00. (Obviously, Screen Clock doesn't set the time for you at system start—you must still do that yourself when you boot the machine or through the DOS commands DATE and/or TIME.)

When you run Screen Clock from DOS, you can select various options by appending commands after typing *CLOCK*. Each command consists of a slash (/), a character, and sometimes a number. Here are the commands and options:

/Cn (Chime), where n is an integer from 0 to 3. /C0 means no chiming; /C1 makes the clock chime hourly; /C2 chimes every half-hour; and /C3 chimes every 15 minutes. A chime is a low beep which lasts for less than one second. Even if the screen updates are turned off, Screen Clock always chimes if you've told it to. The default is no chiming.

/Un (Update), where n is an integer from 1 to 9. This sets how often screen updates are to take place—n is the number of half-seconds between updates. The more frequent the updates, the more often the date and time are refreshed on the screen. However, more frequent updates also make other programs run more slowly. The default is equivalent to /U2 (one second between updates).

/M (Military time). This selects military (24-hour) time. /S (Standard time). This selects standard 12-hour time with an a.m./p.m. marker. Screen Clock defaults to standard time.

/R (Reset). This resets the log-on timer. Screen Clock automatically resets itself to 00:00 when first run.

For example, typing CLOCK /U3/M/C1 at the DOS prompt loads and runs Screen Clock, sets updates every 1½ seconds, sets military time, and makes the clock chime every hour. (Notice that you don't have to enter the optional commands in any particular order, so long as they all follow CLOCK, and are separated by slashes.)

# The Disappearing Clock

Occasionally, the Screen Clock sysline may get in the way. For example, it may hide text printed on the top line of the screen. You can make it disappear by pressing Ctrl and both Shift keys simultaneously. Pressing this combination again turns the sysline back on.

Because Screen Clock maintains its own clock, it may not agree precisely with the DOS clock. Generally, it's never more than half a minute off.

Note that the day, date, and time are reset every time you run Screen Clock. If you change the system date and time, you can reset Screen Clock by running it again. For example, the following would reset the display to 8:00 p.m. on February 14 (the A> prompts are supplied by DOS):

A> time 20:00:00 A> date 02-15-86 A> clock

The log-on time is not reset unless you append the /R command to CLOCK.

Although Screen Clock makes it appear that your computer is doing more than one thing at a time, it's important to remember that computers can really perform only *one* task at a time (a factor of the basic architecture of all personal computers to date). If the computer spends some if its time updating the sysline, that's time away from running the main program. Thus, the more often the sysline is updated, the more time it steals from the computer, and the slower the main program seems to run. However, the part of Screen Clock that takes the most time is printing the sysline on the screen. If screen updates are turned off, there is virtually no slowdown. So during heavy number crunching you might want to turn the sysline updates off.

I've been using Screen Clock quite a bit and haven't noticed much loss of performance at all. It seems that the computer spends a lot of its time waiting for input (from the keyboard, the disk drives, and so on); all we're doing is giving it something else to do in its spare time. I have yet to find a program which doesn't work with Screen Clock.

As the power of personal computers increases, it becomes possible to include features once found only on large mainframe computers. A sysline such as Screen Clock is another

step in this direction.

## How It Works

Mainframe syslines are generally on the bottom row of the screen. The Screen Clock sysline, however, must be on the top row because there's no way via PC-DOS to keep the bottom row from scrolling. The sysline would keep traveling up the screen every time the screen was scrolled. By placing the sysline on the top row, it can be refreshed each time it scrolls off the top of the screen.

The program itself is broken into two sections, resident and nonresident. The resident portion updates the internal counters, sounds the chimes, and updates the screen display. It's driven by the user interrupt 1Ch and is executed about 18 times a second. The nonresident part sets the initial date and

time and changes the program's options.

When you execute CLOCK.COM, the program first checks to see whether the resident portion is already installed. This is important only when the program returns control to DOS. Then it sets the current date and time and checks for any optional parameters. After this, the program is ready to return to DOS. If the program was already installed, it simply returns to DOS and does nothing else. If it needs to be installed, it first deallocates the environment space, then returns to DOS with the "terminate but stay resident" call to store the resident portion of the program safely in memory.

# Screen Clock

For error-free program entry, be sure to use "The Automatic Proofreader," Appendix B.

EF 100 CLS:LOCATE 10,10:PRINT"Writing file ..."
IN 110 OPEN "clock.com" FOR OUTPUT AS #1
EL 120 FOR I=1 TO 1310:READ BYTE:CKSUM=CKSUM+BYTE
:IF BYTE<0 THEN FOR J=1 TO ABS(BYTE):PRINT
#1,CHR\$(0);:NEXT J:GOTO 140</pre>

```
酬 13Ø ₱RINT#1, CHR$(BYTE);
60 140 NEXT I:CLOSE 1
NO 150 IF CKSUM <> 124185! THEN PRINT"** Error in
       DATA statements **":KILL "clock.com":STOP
HJ 160 PRINT:PRINT"File for clock.com has been cr
      eated.": END
HM 200 DATA 233,51,4,74,97,110,32,70,101,98,32,77
KO 216 DATA 97,114,32,65,112,114,32,77,97,121,32,
      74
FF 220 DATA 117,110,32,74,117,108,32,65,117,103,3
      2,83
AD 230 DATA 101,112,32,79,99,116,32,78,111,118,32
      , 68
OF 240 DATA 101,99,32,31,28,31,30,31,30,31,31,30
NA 250 DATA 31,30,31,83,117,110,32,77,111,110,32,
      84
KH 260 DATA 117, 101, 32, 87, 101, 100, 32, 84, 104, 117, 3
      2,70
EK 270 DATA 114,105,32,83,97,116,32,-6,1,0,1,80
EL 280 DATA 19,2,-5,240,18,0,1,-86,13,255,80,97
HC 290 DATA 117,108,80,83,81,82,86,87,85,30,6,140
ED 300 DATA 200,142,216,142,192,232,198,1,232,45,
      Ø,232
BM 310 DATA 133,0,160,108,1,58,6,109,1,114,23,187
CJ 320 DATA 91,1,232,158,1,137,14,95,1,232,144,0
HK 33Ø DATA 128,62,111,1,0,116,3,232,250,0,7,31
N 340 DATA 93,95,94,90,89,91,88,207,180,2,205,22
6C 35Ø DATA 36,7,6Ø,7,116,6,198,6,115,1,Ø,195
EN 360 DATA 128,62,115,1,0,117,67,128,54,111,1,1
IN 370 DATA 198,6,115,1,1,128,62,111,1,0,116,4
M 380 DATA 232, 197, 0, 195, 180, 15, 205, 16, 136, 62, 11
      4,1
NM 390 DATA 180,3,205,16,137,22,112,1,180,2,186,-
IN 400 DATA 205, 16, 185, 31, 0, 176, 32, 180, 14, 205, 16,
      226
PB 41Ø DATA 25Ø,18Ø,2,138,62,114,1,139,22,112,1,2
      Ø5
F 42Ø DATA 16,195,128,62,199,1,255,116,25,160,19
      9,1
LP 430 DATA 58,6,198,1,119,5,254,6,199,1,195,198
KB 44Ø DATA 6,199,1,255,228,97,36,252,23Ø,97,195,
       138
NH 45Ø DATA 22,197,1,128,25Ø,Ø,117,1,195,128,62,9
8B 46Ø DATA 1,Ø,117,8,128,62,196,1,Ø,116,62,195
AD 470 DATA 128,250,1,116,50,128,62,95,1,30,117,8
F6 48Ø DATA 128,62,196,1,0,116,42,195,128,250,2,1
JP 490 DATA 30,128,62,95,1,15,117,8,128,62,196,1
```

```
0A 500 DATA 0,116,22,195,128,62,95,1,45,117,8,128
El 510 DATA 62,196,1,0,116,7,195,198,6,196,1,0
BL 52Ø DATA 195,198,6,196,1,1,198,6,199,1,Ø,176
CH 53Ø DATA 182,23Ø,67,184,1Ø2,1Ø,23Ø,66,138,196,
      230.66
F6 54Ø DATA 228,97,12,3,23Ø,97,195,198,6,1Ø8,1,Ø
N 550 DATA 191,116,1,252,139,54,102,1,209,230,20
      9,230
HM 560 DATA 129,198,63,1,185,4,0,243,164,139,54,9
NL 570 DATA 1,209,230,209,230,129,198,255,0,185,4
      ,ø
FJ 580 DATA 243,164,160,99,1,232,22,1,184,44,32,1
      71
IP 590 DATA 160,101,1,232,12,1,160,100,1,232,6,1
MN 600 DATA 176,32,170,139,14,95,1,138,38,110,1,2
ME 610 DATA 1,1,184,32,40,171,187,104,1,232,58,0
NI 620 DATA 180,1,232,242,0,176,41,170,180,15,205
      , 16
      DATA 136,62,114,1,180,3,205,16,137,22,112,
IA 630
JA 640 DATA 180,2,186,-2,205,16,190,116,1,139,207
      , 43
!! 650 DATA 206,172,180,14,205,16,226,249,180,2,1
      38,62
CF 660 DATA 114,1,139,22,112,1,205,16,195,139,87,
LB 67Ø DATA 139,7,187,69,4,247,243,179,60,246,243
      , 138
10 680 DATA 232,138,204,195,187,91,1,232,16,0,115
BM 690 DATA 232,43,0,187,104,1,232,5,0,254,6,108
JA 700 DATA 1,195,255,7,117,3,255,71,2,131,127,2
NK 710 DATA 24,114,17,129,63,176,0,114,11,199,7,-
KF 72Ø DATA 199,71,2,-2,249,195,248,195,255,6,102
      , 1
FN 730 DATA 131,62,102,1,6,118,6,199,6,102,1,-2
MA 740 DATA 254,6,99,1,139,22,97,1,232,49,0,58
6A 75Ø DATA 22,99,1,115,42,198,6,99,1,1,255,6
OA 760 DATA 97,1,131,62,97,1,12,118,26,199,6,97
JD 770 DATA 1,1,0,254,6,100,1,128,62,100,1,99
ID 780 DATA 118,9,198,6,100,1,0,254,6,101,1,195
PG 79Ø DATA 138,218,5Ø,255,138,151,5Ø,1,128,251,2
      ,117
₩ 800 DATA 16,246,6,100,1,3,117,9,128,62,100,1
D 810 DATA 0,116,2,254,194,195,212,10,5,48,48,13
```

- FK 82Ø DATA 196,171,195,182,32,128,252,1,116,18,1 82,65
- FL 830 DATA 128,253,12,114,5,182,80,128,237,12,10,237
- NL 84Ø DATA 117,2,181,12,138,197,232,217,255,176, 58,17Ø
- OF 850 DATA 138,193,232,209,255,128,254,32,116,3, 138,198
- EN 860 DATA 170,195,82,101,113,117,105,114,101,11 5,32,68
- MA 870 DATA 79,83,32,50,46,48,32,111,114,32,97,98
- DH 880 DATA 111,118,101,46,13,10,36,78,111,119,32
- FL 890 DATA 110,115,116,97,108,108,105,110,103,32,114,101
- IH 900 DATA 115,105,100,101,110,116,32,112,111,11
  4,116,105
- IN 910 DATA 111,110,32,111,102,32,67,76,79,67,75,
- NN 920 DATA 13,10,36,39,32,117,110,107,110,111,11 9,110
- AE 93Ø DATA 32,112,97,114,97,109,101,116,101,114,46,13
- EI 940 DATA 10,36,83,112,101,99,105,102,121,32,97
- NV 950 DATA 110,117,109,98,101,114,32,102,114,111,109,32
- CL 96% DATA 49,45,57,32,102,111,114,32,39,85,39,3
- IC 970 DATA 115,119,105,116,99,104,13,10,36,83,11 2,101
- 6E 98Ø DATA 99,105,102,121,32,97,32,110,117,109,9 8,101
- 6F 99Ø DATA 114,32,102,114,111,109,32,48,45,51,32
- A6 1000 DATA 111,114,32,39,67,39,32,115,119,105,1 16,99
- FB 1010 DATA 104,13,10,36,-2,47,180,48,205,33,60,
- MC 1020 DATA 117,9,186,125,4,180,9,205,33,205,32,
- MD 1030 DATA 0,55,205,33,136,22,53,5,187,125,4,17
- E 1040 DATA 4,211,235,67,137,30,51,5,184,28,53,2
- JI 1050 DATA 33,190,200,1,141,127,252,185,4,0,252
- l 1060 DATA 166,131,249,0,116,41,180,9,186,154,4

- PN 1070 DATA 33,184,28,37,186,204,1,205,33,30,7,2 32
- EN 1080 DATA 32,0,232,87,0,161,44,0,142,192,180,7
- BA 1090 DATA 205,33,184,0,49,139,22,51,5,205,33,2
- EA 1100 DATA 8,0,232,63,0,184,0,76,205,33,6,31
- IK 1110 DATA 180,0,205,26,137,22,91,1,137,14,93,1
- M 1120 DATA 180, 42, 205, 33, 50, 228, 163, 102, 1, 138, 1 98, 163
- JN 1130 DATA 97,1,136,22,99,1,198,6,101,1,19,129
- KA 1140 DATA 233,108,7,128,249,99,118,7,128,233,1 00,254
- PD 1150 DATA 6,101,1,136,14,100,1,195,30,14,31,19
- DN 1160 DATA 129,0,252,172,60,32,116,251,60,13,11 6,51
- HD 1170 DATA 58,6,53,5,116,241,138,224,36,223,60,
- NH 1186 DATA 116,39,60,77,116,51,60,83,116,55,60,
- BN 1190 DATA 116,59,60,67,116,90,80,178,39,180,2, 205
- LB 1200 DATA 33,88,138,212,180,2,205,33,186,198,4
- #D 1210 DATA 9,205,33,31,195,38,199,6,104,1,-2,38
- LF 1220 DATA 199,6,106,1,-2,235,178,38,198,6,110,
- CK 123Ø DATA 1,235,17Ø,38,198,6,11Ø,1,Ø,235,162,1 72
- HH 1240 DATA 60,49,114,21,60,57,119,17,44,48,177,
- F 1250 DATA 138,224,210,228,2,224,38,136,38,109, 1,235
- M 1260 DATA 136,186,221,4,180,9,205,33,235,187,1 72,60
- BK 1270 DATA 48,114,13,60,51,119,9,44,48,38,162,1
- 9 1280 DATA 1,233,109,255,186,8,5,180,9,205,33,2
- HD 1290 DATA 160.0

# Smooth-Scrolling Billboards

Paul W. Carlson

Do you want to leave a message on your computer screen that's sure to be noticed? Or would you like to create an eye-catching display in a shop window that effectively communicates your message to the public? The programs presented here let you easily produce smooth-scrolling billboards on the 40- or 80-column screen of your IBM PC (with color/graphics adapter and BASICA) or PCjr (with Cartridge BASIC).

To be really effective, a billboard program must *smoothly* scroll its message across the screen. Programs that jerk the letters across the screen are very hard on the eyes. The speed necessary for smooth scrolling can be achieved only by avoiding the routines in the BIOS (Basic Input/Output System) and writing directly to video memory. However, this can cause a problem when text is used in graphics modes—writing directly to video memory disrupts the character generator. As a result, small flickering lines appear on the screen (for more details, see COMPUTE! Books's *Mapping the IBM PC and PCjr*, pages 193–198).

This problem can be solved by writing to video memory only during the time when the monitor's raster beam is in vertical retrace, while the display is idle. On some IBM-compatible computers (the Compaq, for example), the problem can be avoided by writing to an inactive page of video memory and then making it the active page. The programs following this article make use of both methods.

With some computer and graphics card combinations, a few flickering lines remain at the very top of the screen when running the 80-column billboard program. These could have been eliminated, but only at the expense of speed and smoothness. About 300 characters can be written to video memory during the vertical retrace period, and 640 characters (eight lines of 80) need to be written for each screen update.

Therefore, to eliminate the flickering lines entirely, we'd have to wait for three vertical retrace periods. These lines are less objectionable than the loss of smoothness caused by waiting for an extra retrace period.

# **Creating Billboards**

Program 1 is for creating billboards on the 40-column screen, and Program 2 is for the 80-column screen. Both programs are extremely easy to use. After typing RUN, simply enter any text string at the prompt. If you want your message to contain a comma, enclose the entire text string in double quotation marks. When you press Enter, the message enlarges and begins scrolling. It can be stopped at any time by pressing the Q key.

The programs can be customized to suit your taste. The character that forms the large letters can be changed from a solid block to another character by changing the DATA statement identified in the listing. For example, to change the solid block to a smiling face, change the DB to 02 in line 300. You can also modify the scrolling speed by changing the two bytes identified in the listing (the second byte has 256 times the effect of the first byte).

## How It Works

The techniques used here can be applied to any program which must update a text screen very rapidly, so a brief description of the steps involved may be useful.

- 1. Set up a buffer in memory equal in size to the block of characters to be written to the screen ( $8 \times 80$  for the 80-column billboard).
- 2. For each input character, access the character PEL map in ROM at FFA6:OE. By columns, depending on whether or not a bit is set, put the code for a solid block or a space into the rightmost column of the buffer array.
- **3.** When a column is complete, scroll the whole buffer one column to the left.
- **4.** Wait for the beginning of a vertical retrace period, then copy the buffer to the inactive screen.
- 5. Make the inactive screen the active screen.
- 6. Do the next column in step 2.

# Program 1. 40-Column Billboards

For error-free program entry, be sure to use "The Automatic Proofreader," Appendix B.

```
EM 10 "
         Forty Column Scrolling Billboard
J0 2Ø 3
LJ 30 1
         Press the "Q" key to quit.
JA 4Ø
DI 50 DEF SEG: CLEAR, &H3FF0: N=&H460A
0L 60 FOR J=0 TO 249: READ A$
BB 70 POKE N+J, VAL ("&H"+A$): NEXT
FA BØ KEY OFF: CLS: SCREEN Ø: WIDTH BØ
6L 9Ø INPUT"Text string":T$:T$=T$+"
FF 100 N=&H4000:K=LEN(T$):FOR J=1 TO K
LP 110 POKE N, ASC (MID$ (T$, J, 1)): N=N+1
04 120 NEXT: POKE N, Ø: CLS: WIDTH 40
JL 130 LOCATE, . Ø: N=&H460A: CALL N
₩ 140 WIDTH 80:CLS: KEY ON: END
8 15Ø DATA Ø6,8B,EC,8C,D8,8E,CØ,B9
8N 16Ø DATA 8Ø,Ø2,8D,3E,Ø8,41,1E,B8
KK 170 DATA 00,B8,8E,D8,BE,30,02,F3
88 180 DATA A4,1F,88,A6,FF,8E,C0,8D
66 190 DATA 36,00,40,8A,1C,46,80,FB
JK 200 DATA 00,74,F4,B7,00,D1,E3,D1
KI 210 DATA E3, D1, E3, 83, C3, ØE, B9, Ø8
FK 220 DATA 00,33,FF,26,8A,07,88,85
8J 23Ø DATA ØØ,41,47,43,E2,F5,56,Ø6

№ 24Ø DATA B9,Ø9,ØØ,51,33,FF,B9,Ø8

FI 250 DATA 00, BB, 4E, 00, D0, A5, 00, 41
MK 26Ø DATA 72,Ø4,BØ,2Ø,EB,Ø2,BØ
IF 270 ' The following value is the
JF 28Ø 3
        ASCII code of character that
ME 290 ' forms the large text.
₩ 3ØØ DATA DB
JH 31Ø DATA 88,87,08,41,83,03,50,47
DN 320 DATA E2,EA,EB,02,EB,B5,8C,DB
CF 33Ø DATA 8E,C3,FC,B8,Ø8,ØØ,8D,36
EK 340 DATA ØA,41,8D,3E,08,41,89,4E
FC 35Ø DATA ØØ,F3,A4,46,46,47,47,48
  36Ø DATA 75,F4,AØ,Ø8,46,34,Ø1,A2
HI 370 DATA 08,46,84,05,50,A8,01,75
60 38Ø DATA Ø5,B8,ØØ,B8,EB,Ø3,B8,8Ø
$E 39Ø DATA B8,8E,CØ,B9,AØ,ØØ,BF,3Ø
HK 400 DATA 02,8D,36,08,41,BA,DA,03
OK 410 DATA EC,A8,08,75,FB,EC,A8,08
60 420 DATA 74,FB,F3,A5,EB,04,EB,8B
JE 43Ø DATA EB, AA, B9, AØ, ØØ, BA, DA, Ø3
OA 44Ø DATA EC,A8,Ø8,75,FB,EC,A8,Ø8
II 450 DATA 74,FB,F3,A5,58,CD,10,B9
FF 460 ' The following two values are the
FJ 470 * time delay constant in the order
```

```
KH 480 ' least sig. byte, most sig. byte. CP 490 DATA 01,00 A6 500 DATA E2,FE,59,E2,DF,07,5E,B4 A6 510 DATA 06,82,FF,CD,21,3C,71,74 NN 520 DATA 06,3C,51,74,02,EB,CF,8B FF 530 DATA E5,07,B8,00,05,CD,10,CB
```

# Program 2. 80-Column Billboards

```
KF 10 *
        Eighty Column Scrolling Billboard
JG 2Ø 3
LJ 3Ø °
        Press the "Q" key to quit.
JA 40 *
DI 50 DEF SEG: CLEAR, &H3FF0: N=&H460A
HC 60 FOR J=0 TO 250: READ A$
B) 70 POKE N+J, VAL ("&H"+A$): NEXT
FA 80 KEY OFF: CLS: SCREEN 0: WIDTH 80
6L 9Ø INPUT"Text string"; T$: T$=T$+"
FF 100 N=&H4000:K=LEN(T$):FOR J=1 TO K
LP 110 POKE N, ASC (MID$ (T$, J, 1)): N=N+1
HD 120 NEXT: POKE N, Ø: CLS
JL 13Ø LOCATE,, Ø: N=&H46ØA: CALL N
CC 140 WIDTH 80:CLS:KEY ON:END
E6 15Ø DATA Ø6,8B,EC,8C,D8,8E,CØ,B9
BH 160 DATA 00,05,8D,3E,08,41,1E,B8
CP 170 DATA 00, B8, 8E, D8, BE, 60, 04, F3
88 180 DATA A4,1F,88,A6,FF,8E,CØ,8D
66 190 DATA 36,00,40,8A,1C,46,80,FB
JK 200 DATA 00,74,F4,B7,00,D1,E3,D1
KI 210 DATA E3, D1, E3, B3, C3, ØE, B9, Ø8
FK 22Ø DATA ØØ, 33, FF, 26,8A, Ø7,88,85
6J 23Ø DATA ØØ, 41, 47, 43, E2, F5, 56, Ø6
DK 240 DATA B9,09,00,51,33,FF,B9,08
KI 250 DATA 00,BB,9E,00,D0,A5,00,41
MK 26Ø DATA 72, Ø4, BØ, 2Ø, EB, Ø2, BØ
JF 270 ' The following value is the
JF 280 ' ASCII code of character that
HE 290 ' forms the large text.
HP 300 DATA DB
GN 31Ø DATA 88,87,Ø8,41,81,C3,AØ,ØØ
JO 320 DATA 47,E2,E9,EB,02,EB,B4,8C
6E 33Ø DATA DB,8E,C3,FC,B8,Ø8,ØØ,8D
ME 34Ø DATA 36,ØA,41,8D,3E,Ø8,41,89
CP 35Ø DATA 9E,ØØ,F3,A4,46,46,47,47
PJ 36Ø DATA 48,75,F4,AØ,Ø8,46,34,Ø1
JD 370 DATA A2,08,46,84,05,50,A8,01
8K 38Ø DATA 75,05,88,00,88,EB,03,88
MH 39Ø DATA ØØ, B9, 8E, CØ, B9, 4Ø, Ø1, BF
```

```
NG 400 DATA 60,04,8D,36,08,41,BA,DA
OF 410 DATA 03,EC,A8,08,75,FB,EC,A8
KO 420 DATA 08,74,FB,F3,A5,EB,04,EB
HC 43Ø DATA BA, EB, AA, B9, 4Ø, Ø1, BA, DA
EL 440 DATA 03,EC,AB,08,75,FB,EC,AB
FI 450 DATA 08,74,FB,F3,A5,58,CD,10
AF 46Ø DATA B9
FH 47Ø *
        The following two values are the
FL 480 ' time delay constant in the order
U 490 ' least sig. byte, most sig. byte.
BO 500 DATA 01,00
91 510 DATA E2, FE, 59, E2, DF, 07, 5E, B4
BI 520 DATA 06, B2, FF, CD, 21, 3C, 71, 74
# 53Ø DATA Ø6,3C,51,74,02,EB,CF,8B
FR 540 DATA E5,07,88,00,05,CD,10,CB
```

# Pie Chart Maker

# Michael Posner

This useful program takes the raw figures you enter (up to nine items) and automatically translates them into percentages to create perfectly proportioned pie charts in color. It requires an Enhanced Model PCjr with Cartridge BASIC or a PC with a disk drive, BASICA, and the color/graphics adapter.

"IBM Pie Chart Maker" uses the medium-graphics screen (SCREEN 1) to create easily understood pie charts. You don't have to be a programmer to use Pie Chart Maker, and a help screen is always available.

If you want to generate a hardcopy printout of a chart, be sure to load the DOS screen-dump utility after booting your system disk (type *GRAPHICS* at the DOS prompt with the DOS disk in the drive, before loading BASIC). Then, to make a screen dump, switch on the graphics printer and press Shift-PrtSc (press Fn-P on the PCjr).

# **Menu Options**

When you run Pie Chart Maker, an option menu appears on the screen:

- 1 Create a pie chart
- 2 Save current chart
- 3 Load chart
- 4 Alter current chart
- 5 Clear current data
- 6 Print chart on screen
- 7 Help
- 8 Exit Pie Chart Maker

To perform one of the functions, press the corresponding number key.

- Option 1 is described in detail in the next section.
- Option 2 (Save current chart) asks you to specify a filename for the chart. When the file is saved, control returns to the menu.

• Option 3 (Load chart) prompts you for the filename of the chart you wish to load. After it's loaded, the menu reappears. Note that loading a chart erases any chart in memory.

• Option 4 (Alter current chart) lets you change data in a chart. Pie Chart Maker lists the current values and asks for the number to be changed. Enter this number, then the new value(s). These are substituted, and again the program asks for the number to be changed. Entering a zero returns you to the menu.

 Option 5 (Clear current data) erases the chart in memory. As a precaution, the program asks for verification before executing this command.

• Option 6 (Print chart on screen) displays the chart on the screen. As mentioned above, use the PrtSc key to reproduce the chart on the printer.

• Option 7 (Help) calls up the help and instructions screen. Press the space bar to return to the menu.

• Option 8 (Exit Pie Chart Maker) quits the program and returns you to BASIC. Again, the program asks for verification.

# Creating a Pie Chart

Creating a pie chart is easy. Let's say you want to chart the annual budget of a small computer company with the money distributed as follows:

Purpose	Amount
1. Research and Development	\$20,000
2. Production	\$18,000
3. Employee wages	\$13,000
4. Advertising	\$11,000
5. Other expenses	\$10,000

The first prompt after selecting option 1 on the menu is *Name of chart?*. An appropriate entry would be *Annual Budget*. For *Number of items?*, you would enter the number 5. Pie Chart Maker accepts up to nine data items.

For *Number 1?*, enter 20,000, and for *Name 1?*, enter *R & D*. Note that names longer than ten characters are shortened to ten in the print-on-screen mode. Enter the other four data items for the budget accordingly.

The program's next prompt asks, Are you using a color monitor (0 = color, 1 = no color)?. Type the appropriate answer. Finally, Pie Chart Maker returns to the menu.

# **Programming Notes**

Pie Chart Maker first computes the percentage of each figure you enter, then draws the main circle. It converts percentages to degrees, then to radians. The BASIC trigonometry functions sine (sin) and cosine (cos) are used to segment the main circle according to the percentages computed for the various data items. As each portion is drawn, it's filled in one of three ways. The PAINT statement of BASIC is the first choice. A second choice is a series of rays from the center of the circle to the edge. The third selection is arcs of decreasing radii. All these may be done in three colors.

Next, the program prints the key at the right of the screen. The circle portions are plotted using a circle command within a FOR-NEXT loop, and then are filled in the same way as the portions of the main circle they represent. Finally, the program prints the percentages and labels.

Line(s)	Function
20-90	Set variables and error trap; go to menu.
100-250	Create chart.
260-270	See if chart is defined.
280-300	Clear variables; draw main circle and initial ray.
310	Sets up main loop; chooses color.
320-350	Find point for line; draw line.
370-420	Fill with PAINT.
440-500	Fill with LINEs from center.
510-570	Fill with arcs.
590-630	Draw circle portions for key.
640-880	Choose filler and fill circle portions.
890-950	Print percentages, name of chart; return to menu.
960-1100	Print options; go to selected option.
1110-1150	Error trapping.
1160-1260	Save chart.
1270-1340	Load chart.
1350-1430	Help and instructions screen.
	Alter data.
1590-1620	Clear chart data.
1630-1660	Exit to BASIC.
1670-1810	Find percentages.

Variables	
Variable(s)	Function
XC	x coordinate of main circle
MR	radius of main circle
XK	x coordinate of circle portions of key
PX	x coordinate for placing percentage
XL	x coordinate for placing data name
CL,SL	point on circle to which ray is drawn
NG\$	name of graph
CV	color: 0=yes, 1=no
N	number of items
XL(N)	datum number n
X(N)	percentage for datum number n
N\$(N)	name for datum number n
YP(N)	coordinate for $n\$(n)$ (percent printed at YP-1)

## IBM Pie Chart Maker

HE 24Ø GOSUB 167Ø

For error-free program entry, be sure to use "The Automatic Proofreader," Appendix B.

```
6C 2Ø KEY OFF: ON ERROR GOTO 111Ø
PK 3Ø XC=99:MR=8Ø:XK=2Ø5:PX=29:XL=3Ø
EO 4Ø FOR X=1 TO 9:READ A:YP(X)=A:NEXT:REM y coor
     dinate for chart I.D.
IA 50 DATA 3,6,8,11,13,16,18,21,23
CA 60 FOR X=1 TO 9:READ Y:C(X)=Y:NEXT:REM color a
     nd type of filler
CE 70 DATA 1,5,9,4,8,3,7,2,6
PP 8Ø PI=3.141593
PO 90 GOTO 960
FG 100 CLS: PRINT TAB (34) "Create chart"
PD 110 PRINT: PRINT
OJ 120 INPUT"Name of chart "; NG$
01 13Ø NG$=LEFT$ (NG$, 26)
KK 140 INPUT"Number of entries (2-9)"; N
IF 15Ø IF N>9 OR N=Ø THEN 1ØØ
J0 16Ø FOR X=1 TO N
MD 17Ø PRINT"Number"X::INPUT X1(X)
PF 18Ø IF X1(X) = \emptyset THEN X1(X) = X1(X-1)
BI 190 PRINT"Name"X;: INPUT N$(X)
OB 200 IF N$(X)="" THEN N$(X)=N$(X-1)
GA 210 NEXT X
KJ 220 INPUT"Are you using a color monitor (0=Y,1
      =N) "; Y$
JN 230 CV=VAL (Y$)
```

```
MG 25Ø RETURN
NC 26Ø IF N<>Ø THEN 28Ø
NC 270 PRINT:PRINT"No chart defined":FOR Z=1 TO 2
      ØØØ: NEXT: RETURN
MA 280 S=0:CD=0:R=0:S1=0:C1=0:CD=0:S2=0:C2=0:CR=0
      :RO=Ø:SP=Ø:D=Ø
HE 290 SCREEN 1,0: IF CV=0 THEN SCREEN 1,1
EL 300 LINE (XC,99)-(XC+MR,99),1
AF 310 CS=0:FOR X=1 TO N:CS=CS+1:CO=C(CS)
BA 320 D=3.6*X(X)+D:R=D*(PI/180):S=SIN(R):C=COS(R
      ): REM find point on circle
IH 330 S1=~((5/6)*MR*S)+99:C1=(MR*C)+XC
NM 340 CIRCLE (XC, 99), MR, CO
N 350 LINE (XC,99)-(C1,S1),CO:REM draw line to p
      oint
BM 36Ø IF CO>3 THEN 43Ø
00 370 REM paint area
J6 38Ø R1=(R-R0)/2+R0:C2=COS(R1)*MR+XC:S2=-(SIN(R
      1) *MR*5/6) +99
JE 390 C2=C2-SGN(C2-XC):S2=S2-SGN(S2-99)
60 400 PAINT(C2, S2), CO, CO
KH 410 CN=C(CS+1):LINE (XC,99)-(C1,S1),CN
ID 420 GOTO 580
CH 43Ø IF CO>6 THEN 52Ø
PE 440 REM lines from center
CK 450 CL=CO-3: FOR A=RO TO R STEP .08
0 460 C3=(COS(A)*MR)+XC:S3=-(SIN(A)*MR*(5/6))+99
86 47Ø LINE (XC.99)-(C3.53),CL
JK 48Ø NEXT A
EB 490 IF CL<3 THEN LINE (XC,99)~(C3,S3),C(CS+1)
IA 500 GOTO 580
IN 510 REM arcs
MG 520 SP=C0-6
CM 53Ø FOR CR=MR TO 1 STEP -7
FM 54Ø IF R>2*PI THEN R=2*PI
KP 550 CIRCLE (XC,99), CR,SP,RO,R
CB 560 NEXT CR
81 570 IF SP<3 THEN LINE (XC,99)-(C1,S1),C(CS+1)
HG 580 RO=R:NEXT X:CIRCLE (XC,99), MR, 3:CIRCLE (XC
      ,99),MR+1.3
DH 590 REM draw circle portions for key
CD 600 FOR C=20 TO (N-1) *20+20 STEP 20
CH 610 CK=C(C/20):CK=CK-(3*INT(CK/3)):IF CK=0 THE
      N CK=3
DM 620 CIRCLE (XK,C),15,CK,-PI/4,-3*PI/4
OI 63Ø NEXT
KB 640 FOR Z=1 TO N
#I 650 CM=C(Z): IF CM<4 THEN GN=1:GOTO 680
OK 660 IF CM>6 THEN GN=3:GOTO 680
CG 67Ø GN=2
```

```
₽ 680 ON GN GOSUB 720,760,840:REM CHOOSE FILLER
      FOR PORTION
OE 69Ø NEXT
FC 7ØØ GOTO 9ØØ
MM 710 REM paint portion
LK 720 CC=CM-INT(CM/3) $3: IF CC=0 THEN CC=3
DM 73Ø YC=15+(2Ø*(Z-1))
CE 74Ø PAINT (XK, YC), CC, CC: GOTO 88Ø
N 750 REM lines from center
LC 760 CC=CM-INT(CM/3) *3: IF CC=0 THEN CC=3
BP 770 YC=20+(20*(Z-1))
MP 780 FOR RC=PI/4 TO 3*PI/4 STEP .3
HM 79Ø CS=COS(RC) *15+XK
EH 800 SC=YC-SIN(RC) *15*5/6
#B 81Ø LINE (XK,YC)-(CS,SC),CC
#I 820 NEXT:GOTO 880
ID 830 REM arcs
LP 840 CC=CM-INT(CM/3) *3: IF CC=0 THEN CC=3
BM 850 YC=20+(20*(Z-1))
09 860 FOR CR=12 TO 2 STEP -3
B 870 CIRCLE (XK, YC), CR, CC, PI/4, 3*PI/4: NEXT
EO 880 YP=YP(Z):LOCATE YP, XL:PRINT LEFT$ (N$ (Z), 10
      ): RETURN
BN 890 REM print percentages
KM 900 FOR Z=1 TO N
聞 910 YP=YP(Z):LOCATE YP~1,PX
LH 920 PRINT USING"##.##"; X(Z):LOCATE YP-1, PX+6:P
      RINT CHR$ (29) "%" * NEXT
HB 93Ø GO=(27-LEN(NG$))/2:LOCATE 2,GO:PRINT NG$:R
      EM print name of graph
NM 940 IF INKEY$ (>" " THEN 940
NN 95Ø RETURN
HF 960 SCREEN 0,0:WIDTH 80:CLS:PRINT TAB(29)"IBM
      Pie Chart Maker"
BJ 980 PRINT: PRINT
PP 990 PRINT TAB(29)"1-Create a pie chart"
M 1000 PRINT TAB(29) "2-Save current chart"
JE 1010 PRINT TAB(29) "3-Load chart"
LA 1020 PRINT TAB(29)"4-Alter current chart"
EB 1030 PRINT TAB(29)"5-Clear current data"
LG 1040 PRINT TAB(29) "6-Print chart on screen"
KC 1050 PRINT TAB(29)"7-Help"
MB 1060 PRINT TAB(29)"8-Exit Pie Chart Maker"
ND 1070 PRINT:PRINT:PRINT"Enter function number:"
LN 1080 F=VAL(INKEY$): IF F<1 OR F>8 THEN 1080
PA 1090 ON F GOSUB 100,1160,1270,1440,1590,260,13
       50,1630
HI 1100 GOTO 960
KC 1110 IF ERL=1300 THEN RESUME 1270
```

- JM 1120 IF ERL=140 THEN PRINT CHR\$(30):RESUME 140
- EA 1130 IF ERL=170 THEN PRINT CHR\$(30):RESUME 170
- NF 1140 IF ERL=1470 THEN PRINT CHR\$(30):RESUME 14
- MA 1150 ON ERROR GOTO 0:END
- FH 1160 SCREEN 0.0:CLS:PRINT TAB(35) "Save chart"
- AB 1170 IF N<>0 THEN 1200
- KN 1180 PRINT"No chart currently defined"
- PF 1190 FOR X=1 TO 1500: NEXT: RETURN
- FC 1200 PRINT: INPUT"Filename to save chart"; NS\$
- 6M 121Ø FOR Z=1 TO LEN(NS\$):IF MID\$(NS\$,Z,1)<>" "
  THEN NEXT:GOTO 123Ø
- AH 1220 PRINT"Please put no spaces in filename.": GOTO 1200
- ME 1230 OPEN NS\$ FOR OUTPUT AS #1
- BF 124Ø PRINT #1,N:PRINT #1,Ns:PRINT #1,CV:PRINT
  #1,NG\$
- GL 1250 FOR S=1 TO N:PRINT #1,X1(S):PRINT #1,N\$(S):NEXT
- N 1260 CLOSE #1:RETURN
- DA 1270 SCREEN 0,0:CLS:PRINT TAB(35) "Load chart"
- DN 1280 PRINT:PRINT:FILES
- LA 1290 PRINT: PRINT: INPUT"Filename of chart": NL\$
- 66 1300 OPEN NL\$ FOR INPUT AS #1
- KB 131Ø INPUT #1,N:INPUT #1,N\$:INPUT #1,CV:INPUT #1,NG\$
- NA 1320 FOR L=1 TO N:INPUT #1,X1(L):INPUT #1,N\$(L):NEXT
- 80 1330 CLOSE #1:GOSUB 1670
- JN 1340 RETURN
- P 135Ø CLS:PRINT TAB(23);"IBM Pie Chart Maker He
  lp & Instructions"
- JP 136Ø PRINT: PRINT" This graph utility makes pie charts from data you provide. From th e menu, press <1> to create a chart. E nter the name of chart, number of items, and theneach data item, along with a name to identify it.
- NO 1370 PRINT: PRINT" Other functions include: saving or loading a chart, altering data, clearing data, printing to screen, or exiting the program."
- #K 138Ø PRINT: PRINT" To perform one of the ab ove functions, press the space bar while your chart is on the screen. This will t ake you to the menu. Then press the numb er of the function you wish to execute.

```
CM 1400 PRINT:PRINT"
                       While your chart is on th
       e screen, hold down SHIFT and press PrtSc
        to send the chart to the printer (press
        Fn-P on the PCjr)."
8J 1410 LOCATE 23,25:PRINT"(Press space bar to co
       ntinue)"
MP 1420 IF INKEY$<>" " THEN 1420
JH 1430 RETURN
KK 1440 CLS:PRINT TAB(35) "Alter data"
NH 1450 PRINT: PRINT: PRINT"# 1 = "NG$" (name of ch
       art)":PRINT"# 2 ="CV;:PRINT"(Ø=color/1=no
        color) ": FOR AD=1 TO N
GB 146Ø PRINT"#"AD+2"="X1(AD)", "N$(AD):NEXT
PQ 1470 PRINT: INPUT "Input # to change (0 to exit)
       ": NC
OF 1480 IF NC=0 THEN 1570: IF NC<1 OR NC>N+3 THEN
       1470
RL 149Ø IF NC>2 THEN 154Ø
OH 1500 INPUT"Enter new data"; NN1$
PO 1510 IF NC=1 THEN NG$=NN1$
BL 1520 IF NC=2 THEN CV=VAL(NN1$)
ON 153Ø GOTO 144Ø
NB 1540 INPUT"Enter new value (number, name)"; NN1
       $. NN2$
BI 1550 NN=VAL(NN1$):X1(NC-2)=NN:N$(NC-2)=NN2$
06 156Ø GOTO 144Ø
01 157Ø GOSUB 167Ø
KN 158Ø RETURN
DL 159Ø SCREEN Ø, Ø: CLS: PRINT TAB(35) "Clear data"
EM 1600 PRINT:PRINT:INPUT"Are you sure";S$
OE 1610 IF S$<>"y" AND S$<>"Y" THEN RETURN
66 162Ø RUN 2Ø
BI 1630 CLS:PRINT TAB(35) "Exit to BASIC"
f! 1640 PRINT:PRINT:INPUT"Are you sure";S$
PA 1650 IF S$<>"y" AND S$<>"Y" THEN RETURN
86 1660 SCREEN 0,0:WIDTH 80:CLS:END
BK 1670 REM compute percentages
C6 168Ø TT=Ø: TA=Ø
*L 1690 SU=0:FOR M=1 TO N:SU=SU+X1(M):NEXT
```

#B 1700 FOR Z=1 TO N:X(Z)=X1(Z)/SU\*100:NEXT

JJ 1740 TA=INT (TT/100): TA=(TA+1) \*100-TT

REM truncate

OL 173Ø IF TT=Ø THEN RETURN

EE 1750 FOR Z=1 TO N:Y(Z)=0:NEXT

: NEXT

EXT

IH 1710 FOR Z=1 TO N:X(Z)=INT(X(Z)\*100)/100:NEXT:

MP 1720 FOR Z=1 TO N; TT=TT+100\*X(Z)-100\*INT(X(Z))

EF 1760 FOR Z=1 TO TA: IF X(Z)<11.11 THEN Y(Z)=1:N

# CHAPTER ONE

```
IP 1770 H=0

OF 1780 H=H+1:IF H>N THEN H=1

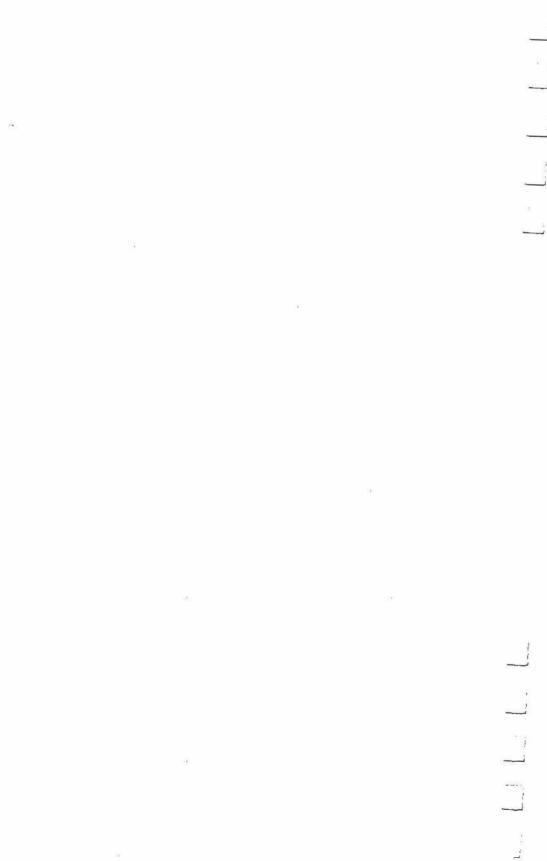
CJ 1790 IF Y(H)=1 THEN 1780

PA 1800 X(H)=X(H)+.01:TA=TA-1:IF TA<=0 THEN RETUR

N

OF 1810 GOTO 1780
```

# CHAPTER TWO Having Fun



# **Switchbox**

Todd Heimarck
PC/PCjr version by Tim Victor

Here's a challenging game of strategy that looks easy at first, but takes time to master and permits many variations. For the IBM PC and PCjr.

Playing "Switchbox" is like putting dominos in place for a chain reaction—either you're setting them in position or you're knocking them over. Winning requires skill and a sense of when to go for points and when to lay back and wait for a better board. The goal is simple: You try to score more points than your opponent by dropping balls into a boxful of two-way switches. Each switch has a trigger and a platform. If the ball lands on an empty platform, it stops dead. But if it hits a trigger, it reverses the switch and continues. In many cases dropping a single ball creates a cascading effect—one ball sets another in motion, which sets others in motion, all the way down.

Type in the program and save a copy before you run it. To play Switchbox on an IBM PC, you need BASICA and a color/graphics adapter; Cartridge BASIC is required if you have a PCjr.

# A Box of Switches

Switchbox is a tale of twos. Each switch has two parts, two positions, two states, two paths in, and two paths out. The two parts are the platform and the trigger. A switch can lean to the left (platform left, trigger right) or to the right (platform right, trigger left); see Figure 1.

The trigger is weak, and always allows balls to pass. But the platform is strong enough to hold a single ball. So the platform either holds a ball—it's full—or it does not and is empty. When a ball sits on a platform, the switch is said to be

loaded, or full.

Figure 1. Trigger States

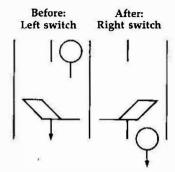


Figure 2. Loaded Trigger

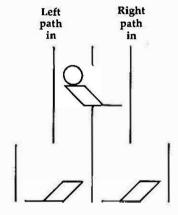


Figure 2 shows a full switch over two empty switches. The platform holds a ball and leans to the left. The trigger extends to the right. Note that the switch on top has two pathways leading in, the left path and the right, and that the right path leading out is the left path into one of the switches below. The left path of the top switch leads into the right path of the other switch below and to the left.

If you drop a ball down the right-hand path, it hits the trigger and flips that switch to the right. Then it continues down, hits the left-hand trigger below and flips that switch as well. In the meantime, the ball on the platform is set in motion (when the switch is flipped) and hits the trigger. The top

switch is reset to point to the left. The second ball then drops a level to the platform below, where it stops.

The playing field in Switchbox is composed of five levels, with four switches in the first level and eight in the bottom level. At the beginning of the game, there are no balls on the field—all platforms are empty—and the position of each switch is chosen randomly.

# Moving Down the Path

Players alternate dropping balls into one of eight entry points. These balls (and others) may or may not make it all the way through the switchbox to one of the 16 exit paths. Balls fall straight down (with one exception), so a ball's movement is always predictable. When it hits an empty switch, one of two things can happen. If it lands on the empty platform, it stops. But if it lands on a trigger, it falls through to the next level below.

Moving balls always make it through loaded switches. Triggers allow balls to continue and move the switch to the other position. If it's loaded, the dead ball on the platform is put into motion, and it hits the trigger that just moved over. This makes the switch go back to its original position, but with an empty platform. So when a ball hits the trigger of a loaded switch, its motion continues unabated. The switch moves, the ball on the platform begins to fall, and it hits the newly placed trigger. The newly emptied switch moves back again, and the two balls drop to the next level.

There's one more possibility: a ball dropping onto a platform that already holds a ball. Since a platform can't hold more than one ball, when this happens, one of the balls slides over to the trigger. The ball doesn't move straight down—it slides over to the next pathway. This is the exception to the rule that balls drop in a straight line. Of course, when the ball hits the trigger, the switch changes position, causing the other ball to drop and hit the trigger.

# The Chain Reaction

At the game's start, all platforms are empty, so four of eight entry paths are blocked. Remember that your turn ends when a ball hits an empty platform and stops. As the switches fill up, the chances increase that a ball will descend through several levels. The goal is to score points by getting balls to pass all the way through the maze of the switchbox. The best way to collect a lot of points is to cause a chain reaction.

A ball that hits a loaded switch from either side continues on its way. The previously inert ball on the platform starts moving. One enters, two exit. If both exiting balls encounter full platforms, four drop from the switches. The pathways are staggered, so the effects can spread outward, with more and more balls cascading toward the bottom.

Rather than taking an easy point or two, it's often worth-while to build up layers of loaded switches. Watch out for leaving yourself vulnerable, though. Because players take turns, you'll want to leave positions where your opponent's move gives you a chance to create a chain reaction. The best strategy is to play defensively. Look ahead a move or two, and watch for an opening that allows you to score several points at once.

# **Four Ouarters**

A game of Switchbox always lasts four rounds. In the first (equality), each exit counts for two points. Your goal is to score ten points. The second quarter has more points available, as well as a higher goal. If you look at the exits, you'll see that the farther away from the middle, the higher the point value. The numbers increase in a Fibonacci sequence: 1, 2, 3, 5, 8, and so on. Each number is the sum of the previous two (1+2 is 3, 2+3 is 5, 3+5 is 8, and so on). The target score in round 2 is 40.

In round 3 the numbers are a bit lower. They increase arithmetically (1, 2, 3, 4, up to 8 in the corners). A goal of 20 points brings you to round 4, where you can score big. Here the numbers are squares: 1, 4, 9, 16, 25, all the way to 64 at the edges. In rounds 2–4, it's sometimes prudent to leave a middle path open for your opponent to score a few points, in order to gather a high score on the big numbers to the left and right.

Each round lasts until one player has reached the specified goal. At that point the other player has one last turn before the round ends. It's possible to win the round on this last-chance play—watch out for barely topping the goal and leaving a chain reaction open for the other player.

As a turn indicator, you'll see an arrow pointing to the scoreboard of the current player. On the other side of the

screen, notice the number where an arrow should be. That's the goal for the current round.

Bonus points are awarded at the conclusion of each round. Four numbers appear below the scorecards. The first is simply the total so far. The second is the total plus a bonus of the goal for the round if the player's points are equal to or greater than the goal. For example, if the goal is 20 and you get 18, there's no bonus. If you score 22, the bonus is the goal for that round (20), and you'd have 42 points. The third number under the scoreboard is the difference between scores for the rounds. If you win by 2 points, 2 is added to your score (and 2 is subtracted from the other player's). The final number is the grand total of the first three scores and bonuses. Rounds 1 and 3 are fairly low-scoring with low goals. You may want to seed the field with extra balls during these quarters, so you can collect more points in the second and fourth quarters.

## **Variations**

Although the goal of the game is to score the most points, there's no reason you couldn't agree to play for low score. In a "lowball" game, you would try to avoid scoring points. You wouldn't necessarily play backward; you would have to adjust the strategy of where to place the balls. Fill up the board as much as possible and leave your opponent in a situation where he or she is forced to score points.

The DATA statements at the beginning of the program determine the goal for each round and the point values for the exit paths. You can prolong the game by doubling the goals. This also dilutes the value of a big score at the beginning of a round, preventing one player from winning on the first or second turn. An interesting variation is to assign negative values to some slots. If some paths score negative points, you're forced to think hard about where the balls will drop.

In addition to the numbered keys (1–8), the plus (+) and minus (-) keys are active. Pressing the plus key drops a ball at random down one of the eight entry paths. Pressing minus allows you to pass your turn to your opponent.

Once you've mastered the regular game, you can add some new rules. Each player gets three passes per half, similar to the three timeouts in a football game. If you don't like the looks of the board, press the minus key to use one of your passes. After one player has skipped a turn, the other player

must play (this prevents the possibility of six passes in a row). It's also a good idea to make a rule that a player can't pass on two consecutive turns. You can also give each player two random moves to be played for the opponent. In other words, after making a move, you could inform your opponent that you're going to give him or her one of your random moves, and you would press the plus key.

Here's one more change you could make: Instead of alternating turns, allow a player to continue after scoring. When a player drops a ball and scores some points, the other player would have to pass (by pressing the minus key). If the first player scores again, the opponent passes again, and so on until no more points are scored.

# **Playing Solitaire**

To drop a ball, press a numbered key (1-8). By using the pass and random turn options, you can play against the computer. Here are the rules for solitaire play:

- 1. The computer always scores first. At the beginning of every round, the computer plays randomly until at least one point is acquired. Press the plus key for the computer's turn. You must continue passing (skip your turn with the minus key) until the computer puts points on the board.
- 2. After the first score by the computer, you can begin to play. When the computer has a turn, press the plus key for a random move.
- 3. Whenever you make points, you must pass again until the computer scores. When the computer gets more points, you can begin to play again. This rule means that you should hold back on the easy scores of a few points; wait until there's an avalanche available.
- 4. If you're the first to reach the goal, the computer gets a last chance. Don't make this move randomly; figure out the best opportunity for scoring and play that move for the last-chance turn.

In the interest of keeping this program to a manageable length, no attempt has been made to provide an "intelligent" computer opponent. Once you become familiar with the game, you might find it an interesting project to try adding some routines that give the computer a rational basis for picking one move over another.

## Switchbox

For error-free program entry, be sure to use "The Automatic Proofreader," Appendix B.

- P8 100 RANDOMIZE TIMER
- LD 110 SCREEN 1,0:CLS
- LO 120 KEY OFF
- 6L 13Ø COLOR 7,Ø
- JB 140 DIM BOX(4,7,1),FALLING(32,4),POINTS(4,16),
  SCORE(1.8)
- A 160 FOR J=1 TO 4: READ POINTS(J,0)
- CK 170 FOR K=1 TO 8:READ L:POINTS(J,K+8)=L:POINTS(J,9-K)=L:NEXT K,J
- 6J 18Ø DATA 10,2,2,2,2,2,2,2,2
- OP 190 DATA 40,1,2,3,5,8,13,21,34
- 0I 200 DATA 20,2,3,4,5,6,7,8,9
- PM 21Ø DATA 8Ø, 1, 4, 9, 16, 25, 36, 49, 64
- BD 22Ø LOCATE 1,1:INPUT "Player 1: ";P1\$:P1\$≃LEFT \$(P1\$,5)
- M 230 INPUT "Player 2: "; P2\$: P2\$=LEFT\$ (P2\$,5)
- KN 24Ø PRINT P1\$;" vs ";P2\$:PRINT "Is this correc
  t? (y/n)"
- PP 250 YN\$=INKEY\$: IF YN\$="" THEN 250
- DA 260 IF YN\$="n" OR YN\$="N" THEN 220
- PJ 27Ø ROUND=4
- PA 280 CLS:GOSUB 970:GOSUB 700
- ES 290 CLS:GOSUB 1030:GOSUB 540
- KP 300 PLAYER=1:PUT (225,1),RARROW:FOR ROUND=1 TO 4:GOSUB 500
- PB 310 FOR I=0 TO 1:CIRCLE (53+256\*I,18+ROUND\*8),
- DL 320 PAINT (53+256\*I,18+ROUND\*8),3:NEXT
- EK 33Ø PLAYER=1-PLAYER:LOCATE 1,9+PLAYER\*21:PRINT SPACE\$(2)
- 0A 34Ø PUT (6Ø,1), LARROW: PUT(225,1), RARROW
- CN 35Ø LOCATE 1,3Ø-PLAYER\*21:PRINT RIGHT\$(STR\$(POINTS(ROUND,Ø)),2)
- DE 360 GOSUB 1210: IF SCORE(1-PLAYER, ROUND) < POINTS (ROUND, Ø) THEN 330
- KO 370 FOR J=0 TO 1:FOR K=5 TO 8
- JI 380 SCORE(J,K)=0:NEXT K,J
- DE 390 FOR J=0 TO 1:FOR K=1 TO 4:BONUS=POINTS(K,0)
  ):AMT=SCORE(J,K)
- #F 400 SCORE(J,5)=SCORE(J,5)+AMT:SCORE(J,6)=SCORE
  (J,6)=BONUS\*(AMT>=BONUS)
- # 410 SCORE(J,7)=SCORE(J,7)+SCORE(J,K)-SCORE(1-J
  ,K):NEXT K,J
- AD 420 FOR J=0 TO 1:FOR K=6 TO 7:SCORE(J,K)=SCORE (J,K)+SCORE(J,5):NEXT K,J

```
JA 430 FOR J=0 TO 1:FOR K=5 TO 7:SCORE(J.8)=SCORE
      (J,8)+SCORE(J,K):NEXT K,J
EK 440 FOR J=0 TO 1:FOR K=5 TO 8:SCORE$=STR$(SCOR
      E(J.K))
HA 45Ø LOCATE K+6, 1+J*34: PRINT SPACE$ (5)
HH 460 LOCATE K+6,5-LEN(SCORE$)+J*34:PRINT SCORE$
      :NEXT K,J
KG 470 NEXT ROUND:LOCATE 11,12:PRINT "Play again?
       (Y/N) "
PD 480 K$=INKEY$: IF K$="n" OR K$="N" THEN CLS: END
      :ELSE IF K$="v" OR K$="Y" THEN RUN: ELSE GO
      TO 48Ø
₩ 500 FOR J=1 TO 16:K=POINTS(ROUND, J):JJ=3+J*2
IF 510 LOCATE 24, JJ: IF K>9 THEN PRINT MID$(STR$(K
      ),2,1); ELSE PRINT " ";
6K 52Ø LOCATE 25, JJ:PRINT RIGHT$ (STR$ (K), 1);:NEXT
NF 53Ø RETURN
@ 540 LINE (4,11)~(54,65),2,BF
NH 550 LINE (9,6)-(59,60),0,BF
IE 56Ø LINE (9,6)-(59,6Ø),1,B
BA 570 LINE (10,18)-(58,18),1
EF 580 GET (4,6)-(59,65),S:PUT (262,6),S,PSET
ML 59Ø LOCATE 2,5-LEN(P1$)/2:PRINT P1$
₽ 600 LOCATE 2,37-LEN(P2$)/2:PRINT P2$
FD 610 FOR J=1 TO 8:LOCATE 1, J*2+10: PRINT J:NEXT
NC 620 FOR SWITCHY=0 TO 4
EH 63Ø FOR SWITCHX=Ø TO SWITCHY+3
AC 640 WP=INT(RND(1)*2):BOX(SWITCHY, SWITCHX, 0)=WP
      :BOX (SWITCHY, SWITCHX, 1) =Ø
#C 65Ø GOSUB 67Ø
M 660 NEXT SWITCHX, SWITCHY: RETURN
BI 670 SY=24+SWITCHY*32:SX=92-SWITCHY*16+SWITCHX*
      32
10 680 IF WP=0 THEN PUT (SX,SY), LEFTSW, PSET ELSE
      PUT (SX,SY), RIGHTSW, PSET
NC 69Ø RETURN
PP 700 FOR I=1 TO 10:LINE (I+155,52)-(I+157,51),2
# 710 LINE (I+189, 20) - (I+191, 19), 2: NEXT
80 720 FOR I=172 TO 180:LINE (I,12)-(I+10,22),1
脚 730 LINE (I+1,11)-(I+3,10),2
LF 740 LINE (1,44)-(I-10,54),1
10 75Ø LINE (I+1,43)-(I+3,42),2:NEXT
HD 760 LINE (186,21)-(200,22),1,BF
NB 770 LINE (166,53)-(154,54),1,BF
80 780 GET (172,6)-(202,22), LEFTSW
WF 790 GET (154,38)-(184,54),RIGHTSW
LE 800 ARC=3.14159/2
NL 810 FOR I=1 TO 2:CIRCLE (80,8), I*4,3,ARC, ARC*2
```

6D 82Ø CIRCLE (68,6),I\*4,3,ARC\*3,ARC\*4 0C 83Ø CIRCLE (231,8),I\*4,3,Ø,ARC

```
0P 840 CIRCLE (243,6), I*4,3, ARC*2, ARC*3: NEXT
#P 850 LINE (80,1)-(88,4),3,BF
PI 860 LINE (231,1)-(223,4),3,BF
FB 87Ø LINE (61,9)~(67,13),3,BF
HL 88Ø LINE (25Ø,9)-(244,13),3,BF
B6 89Ø PAINT (74,7),3
1 900 PAINT (237,7),3
#D 910 FOR I=5 TO 17:LINE (58,11)-(64,1),3
EH 920 LINE (253,11)-(247,1),3:NEXT
IL 93Ø GET (58,1)-(88,17), LARROW
F 940 GET (223,1)-(253,17), RARROW
M 95Ø RETURN
# 960 '---- DRAW SWITCHBOX -----
IM 970 CIRCLE (100,100),3,3
E 980 PAINT (100,100),3
WE 990 GET (97,97)-(103,103), BALL
AH 1000 PUT (97,97), BALL, PRESET
HL 1010 GET (97,97)-(103,103), UNBALL
IB 1020 RETURN
SB 1030 LINE (80,24)-(87,39),1,BF
MC 1040 LINE (224,24)-(231,39),1,BF
HH 1050 FOR I=0 TO 7:LINE (81+1,23)-(96+1,8),1
PM 1060 LINE (208+1,8)-(223+1,23),1:NEXT
PD 1070 GET (80,8)-(103,39),S
HP 1080 FOR I=0 TO 3:PUT (64-I*16, I*32+40), S:NEXT
HK 1090 GET (208,8)-(231,39),S
JE 1100 FOR I=0 TO 3:PUT (224+1*16, I*32+40), S:NEX
       T
LK 1110 LINE (96,8)-(215,15),0,BF
N 1120 LINE (16,168)-(23,183),1,BF
KI 1130 LINE (288, 168) - (295, 183), 1, BF
NG 1140 FOR I=O TO 7:LINE (16+I,184)-(29,197-I),1
KK 1150 LINE (288+I, 184) - (282, 190+I), 1:NEXT
BF 1160 FOR I=0 TO 6:FOR HP=123-I*16 TO 187+I*16
       STEP 32
MF 1170 VS=I*32-32: VE=VS+64: IF VS<8 THEN VS=8
ME 1180 IF VE>186 THEN VE=186
DK 1190 LINE (HP, VS) - (HP, VE), 1: NEXT: NEXT
IP 1200 RETURN
IK 1210 'GAME STUFF
FD 1220 FOR FBALL=0 TO 32: FALLING (FBALL, 0) =0: NEXT
       : NEWBALL=1
LA 1230 A$="":WHILE A$="":A$=INKEY$:WEND
GE 1240 IF A$="-" THEN RETURN
PK 1250 IF A$="+" THEN A$=CHR$(INT(RND(1) *8+49))
EI 1260 A=VAL(A$): IF A<1 OR A>8 THEN 1230
#K 127Ø FALLING(Ø,Ø)=1:FOR J=1 TO 3:FALLING(Ø,J)=
       Ø: NEXT
88 128Ø FALLING(Ø, 4)=1Ø+A*2
```

PL 1290 EXIT=0:WHILE EXIT=0:EXIT=1

# CHAPTER TWO

- OH 1300 FOR FBALL=0 TO 32:IF FALLING(FBALL,0)=1 T HEN EXIT=0:GOSUB 1320
- AA 1310 NEXT: WEND: RETURN
- FM 132Ø DY=FALLING(FBALL,0):DX=FALLING(FBALL,1):L
  EVEL=FALLING(FBALL,2)
- HA 1330 NY=FALLING(FBALL, 3): NX=FALLING(FBALL, 4)
- IH 1340 IF LEVEL<>0 OR NY<>0 THEN GOSUB 1570
- AB 1350 NY=NY+1: FALLING (FBALL, 3)=NY AND 3:0N NY G
  OTO 1360,1380,1420,1430
- FK 136Ø IF LEVEL≂5 THEN FALLING(FBALL,Ø)=Ø:GOTO i 500
- ## 137Ø GOSUB 156Ø:ON INT(RND(1)\*3+1) GOTO 158Ø,1
  59Ø,16ØØ
- # 1380 VX=0:GOSUB 1540
- LC 1390 IF BOX(SWITCHY,SWITCHX,1)=1 AND BOX(SWITC HY,SWITCHX,0)=SIDE THEN VX=1-2\*SIDE:FALLI NG(FBALL,1)=VX:NX=NX+VX:FALLING(FBALL,4)= NX:GOSUB 1560:GOTO 1610
- M 1400 GOSUB 1560:IF BOX(SWITCHY,SWITCHX,0)=SIDE THEN FALLING(FBALL,0)=0:BOX(SWITCHY,SWIT CHX,1)=1:GOTO 1620
- PA 1410 ON INT(RND(1) \*3+1) GOTO 1580,1590,1600
- 16 142Ø FALLING(FBALL,1)=Ø:NX=NX+DX:FALLING(FBALL
  ,4)=NX:GOSUB 156Ø:GOTO 163Ø
- #E 1430 FALLING(FBALL, 2)=LEVEL+1:GOSUB 1560
- CL 144Ø GOSUB 154Ø:BOX(SWITCHY,SWITCHX,Ø)=1~BOX(S WITCHY,SWITCHX,Ø)
- DB 1450 IF BOX(SWITCHY, SWITCHX, 1) = 0 THEN 1490
- # 1460 FALLING(NEWBALL,0)=1:FALLING(NEWBALL,1)=0
  :FALLING(NEWBALL,2)=LEVEL
- 6L 147Ø FALLING(NEWBALL,3)=Ø:FALLING(NEWBALL,4)=N
  X+2~SIDE\*4
- #D 148Ø BOX(SWITCHY,SWITCHX,1)≃Ø:NEWBALL=NEWBALL+ 1:GOSUB 164Ø
- 00 1490 WP=BOX(SWITCHY,SWITCHX,0):GOSUB 670:GOTO
- M 1500 AMT=POINTS(ROUND,NX/2-1):SUBTOT=SCORE(PLA
  YER,ROUND)+AMT
- N 1510 SUBS=STRS(SUBTOT):LOCATE ROUND+3,7-LEN(SUBS)+PLAYER\$32:PRINT SUBS
- #0 1520 SCORE(PLAYER, ROUND) = SUBTOT
- AC 153Ø GOTO 165Ø
- !! 1540 SWITCHY=LEVEL:JX=NX/2+LEVEL-5
- JL 1550 SWITCHX=INT(JX/2):SIDE=JX-INT(JX/2)\*2:RET URN
- 00 1560 PUT (NX\*8,8+LEVEL\*32+NY\*8), BALL, OR: RETURN
- 0E 157Ø PUT (NX\*8,8+LEVEL\*32+NY\*8),UNBALL,AND:RET
   URN
- # 1580 FOR I=0 TO 1:SOUND 880,1:SOUND 32767,1:NE
   XT:RETURN

# Having Fun

- KL 159Ø FOR I=Ø TO 1:SOUND 66Ø,1:SOUND 32767,1:NE XT:RETURN
- CH 1600 FOR I=0 TO 1:SOUND 440,1:SOUND 32767,1:NE XT:RETURN
- 00 1610 FOR I=1 TO 6:SOUND 1100\*RND(1)+37,1:NEXT: RETURN
- IM 1620 FOR I=800 TO 200 STEP -20:SOUND I,.1:NEXT :RETURN
- GA 163Ø FOR I=1 TO 5:SOUND 55Ø\*RND(1)+37,1:NEXT:R ETURN
- FB 1640 FOR I=0 TO 1:FOR J=440 TO 880 STEP 80:SOU ND J..5:NEXT J.1:RETURN
- KJ 1650 FOR I=0 TO 5:SOUND 330,.5:SOUND 440,.5:SO UND 550,.5:NEXT
- KI 1660 SOUND 32767, 1: RETURN

# Webster Dines Out

Walter Bulawa PC/PCjr version by Charles Brannon

Tired of blasting invaders from outer space? This whimsical game is set in a very different world—the miniature jungle in your own backyard.

Guide Webster, the hungry tree spider, in his endless search for a square meal. Roving back and forth across his tree limb, he watches for bugs to appear in the grass below. When the time is right, he drops down on a strand of silk for a light snack, then climbs back up his web to look for more.

Unfortunately, this backyard paradise isn't quite perfect. The more Webster eats, the faster the bugs move, making it harder to find the next meal. Even worse, he's not the only one with an appetite—there's a speedy scorpion sharing the same hunting ground, stealing bugs when he can and giving Webster a sting whenever he drops too close.

Drop from the Glistening Thread

"Webster Dines Out" runs on any PCjr with Cartridge BASIC and on any IBM PC with BASICA and a color/graphics adapter.

To move Webster, press the left and right cursor keys. When a bug passes below, hit the space bar to drop Webster to the ground.

Webster's goal in life is to score points as quickly as possible. Keep an eye on his lifeline at the top of the screen, though. When Webster drops to get a bug, his energy level is drained and the lifeline shrinks. Capturing a bug restores his energy and expands the lifeline.

Scoring is simple—10 points for catching a beetle, 20 for each bug, and bonus points for multiple captures. Extra bonus points are awarded at the 1,000, 5,000 and 10,000 point marks. As the score increases, the bugs speed up and become scarcer. Webster's energy drains faster, too. The game ends when he hits the scorpion or his energy is depleted.

#### Webster Dines Out

For error-free program entry, be sure to use "The Automatic Proofreader," Appendix B.

- 16 1 'Webster Dines Out for IBM PC with Color/Graphics Adaptor and BASICA or Expanded PCjr
- IG 2 PLAY "mb
- E 100 DEFINT A-Z:KEY OFF:WIDTH 40:SCREEN 0,1,0:C OLOR 7,0,0:CLS:RANDOMIZE TIMER
- NO 110 FOR I=1 TO 11:COLOR, I:LOCATE I,1,0:PRINT SPACE\$(39):COLOR, 15-I:LOCATE 24-I,1:PRINT SPACE\$(39):NEXT
- F) 120 LOCATE 12,1:COLOR 0,15:PRINT SPACE\$(12)+"W ebster Dines Out"+SPACE\$(10)
- 00 125 LOCATE 16,1:PRINT" Do you want sound
  7 (Y/N)";:A\$=INPUT\$(1):IF A\$="y" OR A\$="Y"
  THEN SOUNDFLAG=1
- NJ 130 IF SOUNDFLAG THEN PLAY "11602cccd cfffg faaa ba>18d 116 ggged ccc<ag fff dc 14<f
- KF 135 GOSUB 55Ø
- 6A 15Ø SCREEN 1:CLS:COLOR 1,Ø:LINE (Ø,18Ø)-(319,1 99),1,BF
- NK 16Ø LINÉ (25Ø,18Ø)-(319,18Ø),2:LINE (25Ø,18Ø)(26Ø,175),2:LINE -(28Ø,15Ø),2:LINE -(283,1
  3Ø),2:LINE -(285,8Ø),2:LINE -(283,5Ø),2:LI
  NE -(27Ø,2Ø),2:LINE -(Ø,2Ø),2::LINE (31Ø,5
  Ø)-(285,13),2:LINE (31Ø,5Ø)-(31Ø,13),2:LIN
  E (Ø,13)-(285,13),2
- N 161 LINE (310,38)-(295,13),2:LINE (310,28)-(30 0,13),2:LINE (310,37)-(310,30),0:LINE (295 ,13)-(300,13),2:LINE (310,13)-(319,13),2:P AINT (0,14),2,2:PSET (303,28),1
- N 162 A\$="c113uIuluuuurdrdrdrdrdrllulu":DRAW A\$:
   PSET (272,29),1:DRAW A\$:PSET (278,40),1:DR
   AW A\$:FOR I=1 TO 5:PSET (250\*RND(1),15):DR
   AW A\$:NEXT
- ## 165 FOR I=Ø TO 15 STEP 2:CIRCLE (300,100),I,Ø,,8/5:NEXT
- DB 17Ø BEETSPEED=2:BEET2SPEED=2:BUGSPEED=2:BUG2SP
  EED=2:SCORPSPEED=4
- LC 180 BEETSCORE=10:BUGSCORE=20:MISSLIFE=10:BUGLI FE=10:BEETLIFE=15:DRAINLIFE!=.5:PROBABILIT Y=20
- KC 19Ø WEBX=1ØØ:WEBY=21:PUT (WEBX,WEBY),WEB,PSET: LIFE!=18Ø:GOSUB 1Ø3Ø:LOCATE 1,1:PRINT"Scor e:";Ø
- OK 200 LIFÉ!=LIFE!-DRAINLIFE!:SOUND 40,.1\*SOUNDFL AG:GOSUB 1030

- NP 21Ø IF SCORPFLAG THEN PUT (SCORPX,SCORPY),SCOR
  P,PSET:SCORPX=SCORPX-SCORPSPEED:IF SCORPX<
  Ø THEN SCORPFLAG=Ø:PUT (SCORPX+SCORPSPEED,
  SCORPY).SCORP</pre>
- ME 22Ø IF BEETFLAG THEN PUT (BEETX,BEETY),BEET,PS
  ET:BEETX=BEETX+BEETSPEED:IF BEETX>231 THEN
  BEETFLAG=Ø:PUT (BEETX-BEETSPEED,BEETY),BE
  ET
- %F 23Ø IF BEET2FLAG THEN PUT (BEET2X,BEET2Y),BEET
  2,PSET:BEET2X=BEET2X-BEET2SPEED:IF BEET2X
  Ø THEN BEET2FLAG=Ø:PUT (BEET2X+BEET2SPEED,
  BEET2Y),BEET2
- CH 240 IF BUGFLAG THEN PUT (BUGX, BUGY), BUG, PSET: B
  UGX=BUGX-BUGSPEED: IF BUGX<Ø THEN BUGFLAG=Ø
  :PUT (BUGX+BUGSPEED, BUGY), BUG
- M 25Ø IF BUG2FLAG THEN PUT (BUG2X,BUG2Y),BUG2,PS
  ET:BUG2X=BUG2X+BUG2SPEED:IF BUG2X>226 THEN
  BUG2FLAG=Ø:PUT (BUG2X~BUG2SPEED,BUG2Y),BU
  G2
- AG 260 K\$=1NKEY\$: IF K\$="" THEN 420
- FG 270 IF K\$=CHR\$(0)+CHR\$(75) THEN PUT (WEBX, WEBY), WEB: WEBX=WEBX+10\*(WEBX>0): PUT (WEBX, WEBY). WEB: GOTO 200
- # 28Ø IF K\$=CHR\$(Ø)+CHR\$(77) THEN PUT (WEBX,WEBY),WEB:WEBX=WEBX-1Ø\*(WEBX<22Ø):PUT (WEBX,WEBY),WEB:GOTO 2ØØ</pre>
- ED 290 IF K\$<>" " THEN 420
- PL 300 FOR I=WEBY TO 145 STEP 3:PUT (WEBX,I), WEB, PSET
- U 31Ø SOUND 5ØØ+I\*2Ø,.1\*SOUNDFLAG
- KP 320 NEXT: WBX=WEBX+15: NUMHIT=0
- H 33Ø IF BEETFLAG THEN IF ABS(WBX-(10+BEETX-BEET SPEED))<1Ø THEN PUT (BEETX-BEETSPEED, BEETY ),BEET:BEETFLAG=Ø:LIFE!=LIFE!+BEETLIFE:SCO RE!=SCORE!+BEETSCORE:GOSUB 97Ø:NUMHIT=NUMH IT+1
- EK 340 IF BEET2FLAG THEN IF ABS(WBX-(10+BEET2X+BE ET2SPEED))<10 THEN PUT (BEET2X+BEET2SPEED, BEET2Y), BEET2: BEET2FLAG=0:LIFE!=LIFE!+BEET LIFE:SCORE!=SCORE!+BEETSCORE:GOSUB 970:NUM HIT=NUMHIT+1
- 81 35Ø IF BUGFLAG THEN IF ABS(WBX-(12+BUGX+BUGSPE ED))<1Ø THEN PUT (BUGX+BUGSPEED,BUGY),BUG: BUGFLAG=Ø:LIFE!=LIFE!+BUGLIFE:SCORE!=SCORE !+BUGSCORE:GOSUB 97Ø:NUMHIT=NUMHIT+1
- NJ 360 IF BUG2FLAG THEN IF ABS(WBX-(12+BUG2X-BUG2 SPEED))<10 THEN PUT (BUG2X-BUG2SPEED, BUG2Y ),BUG2:BUG2FLAG=0:LIFE!=LIFE!+BUGLIFE:SCOR E!=SCORE!+BUGSCORE:GOSUB 970:NUMHIT=NUMHIT +1

- MN 37Ø IF SCORPFLAG THEN IF ABS(WBX-(15+SCORPX+SC ORPSPEED))<1Ø THEN WEBY=145:GOTO 107Ø
- F) 38Ø IF NUMHIT=Ø THEN LIFE!=LIFE!-MISSLIFE:IF L IFE!<1 THEN WEBY=145</p>
- FL 390 GOSUB 1030: IF NUMHIT>1 THEN SCORE!=SCORE!+
  10^NUMHIT:GOSUB 980:FOR W=1 TO 50:SOUND 30
  00+10\*(W AND 1),.05\*SOUNDFLAG:NEXT:LINE (0
  ,170)-(249,179),0,BF
- M6 400 FOR I=140 TO WEBY STEP-3:PUT (WEBX,I), WEB, PSET:SOUND I\*20+500,.1\*SOUNDFLAG:NEXT:PUT (WEBX,WEBY), WEB, PSET
- CG 405 IF INKEY\$<>"" THEN 405
- BL 41Ø GOTO 2ØØ
- 10 42Ø IF 100\*RND(1)>PROBABILITY THEN 200
- BK 43Ø ON 5\*RND(1)+1 GOSUS 44Ø,46Ø,48Ø,5ØØ,52Ø:GO TO 2ØØ
- FC 44Ø IF BEETFLAG=Ø THEN BEETFLAG=1:BEETX=Ø
- NI 45Ø RETURN
- KA 460 IF BEET2FLAG=0 THEN BEET2FLAG=1:BEET2X=231
- NN 47Ø RETURN
- FD 48Ø IF BUGFLAG=Ø THEN BUGFLAG=1:BUGX=226
- NA 49Ø RETURN
- CF 500 IF BUG2FLAG=0 THEN BUG2FLAG=1:BUG2X=0
- MB 51Ø RETURN
- JL 520 IF SCORPFLAG=0 THEN SCORPFLAG=1:SCORPX=218
- NF 53Ø RETURN
- L6 54Ø END
- 13 55Ø BOTSCR=18Ø
- PK 56Ø READ X,Y:E=(4+INT((X+7)/8)\*Y)/2:DIM WEB(E)
  :WEB(Ø)=X:WEB(1)=Y:WEBY=BOTSCR-Y:FOR I=2 T
  0 E:READ WEB(I):NEXT
- 0H 57Ø READ X,Y:E=(4+INT((X+7)/8)\*Y)/2:DIM BEET(E
  ):BEET(Ø)=X:BEET(1)=Y:BEETY=BOTSCR-Y:FOR I
  =2 TO E:READ BEET(I):NEXT
- FK 58Ø READ X,Y:E=(4+INT((X+7)/8)\*Y)/2:DIM BEET2(
  E):BEET2(Ø)=X:BEET2(1)=Y:BEET2Y=BOTSCR-Y:F
  OR I=2 TO E:READ BEET2(I):NEXT
- #C 59Ø READ X,Y:E=(4+INT((X+7)/8)\*Y)/2:DIM BUG(E)
  :BUG(Ø)=X:BUG(1)=Y:BUGY=BOTSCR-Y:FOR I=2 T
  @ E:READ BUG(I):NEXT
- EA 600 READ X,Y:E=(4+INT((X+7)/8)\*Y)/2:DIM BUG2(E
  ):BUG2(0)=X:BUG2(1)=Y:BUG2Y=BOTSCR-Y:FOR I
  =2 TO E:READ BUG2(I):NEXT
- CD 61Ø READ X,Y:E=(4+INT((X+7)/8)\*Y)/2:DIM SCORP(
  E):SCORP(Ø)=X:SCORP(1)=Y:SCORPY=BOTSCR-Y:F
  OR I=2 TO E:READ SCORP(I):NEXT
- ME 62Ø RETURN
- LF 63Ø END
- 10 640 DATA &H3C,&H1a,&H0,&H500,&H0,&H0,&H0,&H500
- E! 650 DATA &HØ, &HØ, &HØ, &H500, &HØ, &HØ, &HØ, &HFF00

- CS 66Ø DATA &HFØ,&HØ,&HØ,&HFFØF,&HFF,&HØ,&HØ,&HBF ØA
- HE 670 DATA &HEA,&HØ,&HØ,&HBFØA,&HEA,&HØ,&HØ,&HBF
- LB 680 DATA &HEB,&HØ,&HØ,&H3FØØ,&HCØ,&HØ,&HØ,&HFF ØF
- GL 69Ø DATA &HCØFF,&HØ,&HØ,&HFFFF,&HFCFF,&HØ,&H57 Ø1.&HF5DF
- HC 700 DATA &H7DFF, &H54, &HF14, &HD5F5, &HFF75, &H400 1, &HF40, &H75FF
- JP 710 DATA &HFFDF, &H10000, &HSF01, &H55F7, &HFF5D, &H54, &H314, &HF55F
- MO 72Ø DATA &HSCFF, &H4ØØ1, &H154Ø, &HD5FF, &HF57F, &H 1Ø4Ø, &H4ØØ1, &HD5FF
- EL 73Ø DATA &HFØ7F,&H14,&H1Ø4,&HF543,&H14FC,&H1,& H141Ø,&HØ
- FI 74Ø DATA &H1ØØ,&H4Ø4Ø,&H4ØØØ,&HØ,&HØ,&H1Ø,&H1, &HØ
- P 760 DATA %H26,%H5,%H0,%HABAA,%HA0,%HAA2A,%H5BA A,%HAA00
- LI 770 DATA &HABAA,&HAØ,&H1101,&H10,&H0,&H0,&H0,&H0,&
- EL 78Ø DATA %H24,%H5,%HAA28,%HAB,%H96ØØ,%HAAAA,%H 8Ø,%HAA28
- IJ 790 DATA &HAØAA, &HØ, &HØ, &HØ, &HØ, &HØ, &HØ, &HØ
- ₩ 800 DATA &H30,&HA,&H3,&H0,&H0,&H300C,&H0,&H0
- NI 810 DATA &HC005, &H0, &H0, &H4395, &HFFFF, &HC0, &H5 F5A, &HFFFF
- F! 820 DATA %HF4,%H55A,%H5555,%H55,%H900,%H5A5A,% H54.%H2000
- IO 830 DATA &H2020,&H0,&H2000,&H2020,&H0,&H0,&H0, &H0
- NE 840 DATA &HØ
- KB 850 DATA &H30,&HA,&H0,&H0,&HC000,&H0,&H0,&H300
- 04 860 DATA &HØ, &HØ, &H5003, &H300, &HFFFF, &H56C1, &H 1F00, &HFFFF
- 10 870 DATA &HASFS, %HSS00, %HSSSS, &HASS0, &H1S00, &H ASAS, &H60, &H0
- KG 880 DATA &H808, &H8, &H0, &H808, &H8, &H0, &H0, &H0
- 0M 89Ø DATA &HØ
- LL 900 DATA &H40, &HD, &H0, &H0, &H2A00, &H80, &H0, &H0
- ON 910 DATA &HA202,&HA0,&H0,&H0,&H8202,&HA0,&H80C,&H0
- #D 920 DATA &H20A,&H80,&H2633,&H0,&HA,&HC0,&HE600 ,&HA882
- F 93Ø DATA &H32A, &HØ, &HA63Ø, &HAAAA, &HA8, &HØ, &HAA C3, &HAAAA

- KI 940 DATA &HAØ, &HØ, &H2A3C, &HAA2A, &HBØ, &HØ, &HØ, & HAAØA
- PF 95Ø DATA &HBØ,&HØ,&HØ,&H1111,&HØ,&HØ,&HØ,&H444
- CL 960 DATA &HØ, &HØ, &HØ, &HØ, &HØ, &HØ, &HF40
- IN 970 LOCATE 1,1:PRINT"Yummy!":SOUND 110,2\*SOUND FLAG:FOR I=600 TO 500 STEP-2:SOUND I,.1\*SO UNDFLAG:NEXT I
- JE 980 LOCATE 1,1:PRINT"Score: ";SCORE!
- JB 99Ø IF SCORE!>5ØØ THEN BUGSPEED=3:BUG2SPEED=3:
  SCORPSPEED=5:PROBABILITY=18:DRAINLIFE!=.6:
  MISSLIFE=15
- FA 1000 IF SCORE!>1000 THEN BUGSPEED=4:BUGSPEED= 4:BEETSPEED=3:BEET2SPEED=3:SCORPSPEED=6:P ROBABILITY=17:DRAINLIFE!=1
- LL 1010 IF SCORE!>5000 THEN BEETSPEED=4:BEET2SPEE
  D=4:DRAINLIFE!=2:PROBABILITY=15:MISSLIFE=
  20
- 18 1020 RETURN
- 36 1030 IF LIFE!>180 THEN LIFE!=180
- # 1040 LINE (136,5)-(136+LIFE!,5),-2\*(LIFE!<30)3\*(LIFE!>30)+2\*(LIFE!>=60):LINE (137+LIFE!,5)-(319,5),0
- N 1050 IF LIFE!<1 THEN 1110
- JN 1969 RETURN
- EA 1070 FOR J=1 TO 5
- 16 1090 FOR I=130 TO 145 STEP 3:PUT (WEBX,I),WEB, PSET:NEXT:SOUND 60,SOUNDFLAG
- 6M 1100 NEXT J
- AF 1110 FOR I=1 TO 10:PUT (WEBX+5\*RND(1)+5\*RND(1)
  \*(WEBX>5),WEBY+5\*RND(1)-5\*RND(1)),WEB:SOU
  ND 40,.1\*SOUNDFLAG:NEXT
- ND 1120 IF INKEY\$<>"" THEN 1120
- JL 1130 LOCATE 13,13:PRINT"Play Again? (Y/N)";:A\$
  =INPUT\$(1):IF A\$="y" OR A\$="Y" THEN RUN
- HH 1140 SCREEN 0,0,0
- HC 115Ø END

# The Witching Hour

Brian Flynn

This game of skill and foresight is ideal for a bleak, stormy October night. For the PCjr with Cartridge BASIC and for any IBM PC with BASICA and a color/graphics adapter.

When autumn winds send a shiver down your spine and the witching hour draws near, there's no better entertainment than a good computer game. "The Witching Hour" is an absorbing contest of strategy based on Alquerque, a board game played in ancient Egypt and still popular in Spain today.

### Witches 1, Ghosts 0

The Witching Hour pits broomstick-straddling witches against ethereal ghosts and is played on a board of 25 squares with 12 pieces to a side. After choosing sides, you attempt to take your opponent's players by jumping over them. You can move vertically, horizontally, or diagonally. However, certain diagonal moves are illegal (the lines between squares show where you can go), and only one square is vacant when the game begins.

Jumping an opposing player's piece removes that piece from the board. If no capture is possible, you may move any piece to an adjacent empty square. You may not pass up a capture—if it's possible to jump an opponent, you *must* always do so—and if the first capture puts you in position to make another, you must jump again. Don't fret—the computer won't let you make illegal moves,

Play ends when all the pieces from one side have been removed from the board. You can play against a friend, measure your skills against the computer, or even watch the computer play itself in a dual-personality flurry.

Like other contests of strategy, The Witching Hour is simple to learn, but a challenge to master, and can be played at many different levels.

Hint: It's sometimes smart to sacrifice a player to draw the opponent into a dangerous position.

# **Ephemeral Elements**

Each game square on the screen is marked with one of the letters of the alphabet. To move a piece, first type the letter for the square of the piece you want to move, then type the letter of the square where you want the piece to go. For instance, to move a witch from square L to square M, type L when the computer prompts you with FROM, and type M when it prompts you with TO. If you press Enter without typing a letter, the computer takes that turn. Thus, to play alone against the computer, just press Enter every other turn. Press Enter on every turn to watch the computer play against itself.

### The Witching Hour

rn":GOTO 300

```
NK 1Ø GOSUB 53Ø:GOTO 28Ø
06 20 H=0:K=0:FOR A=7 TO 35:GOSUB 40:NEXT
8 30 GOSUB 170: IF H<1 THEN 250
6E 4Ø H=Ø:K=Ø:A=T:GOSUB 6Ø:IF H<1 THEN 25Ø
HI 50 GOTO 30
DK 60 IF B(A)=0 OR B(A)=-S OR B(A)=2 THEN RETURN
# 70 FOR B=0 TO D(A-7):C=A+M(B):IF B(C)=S OR B(C
     )=2 THEN 16Ø
OC BØ IF B(C) THEN 12Ø
ND 90 SC=RND(1)*.9:IF H<SC THEN H=SC:F=A:T=C
AE 100 IF CK=1 AND T1=C THEN L=1:B=7
EA 11Ø GOTO 14Ø
NN 120 IF B(C+M(B)) THEN 160
JN 130 SC=1+RND(1)*.9:IF H<SC THEN H=SC:F=A:T=C+M
      (B):K=C
MK 14Ø IF CK=Ø THEN 16Ø
SP 150 IF T1=C+M(B) THEN L=1:K1=C:B=7
SN 160 NEXT: RETURN
柳 170 B(T)=B(F):B(F)=Ø:A=F:GOSUB 760
IF 180 IF K THEN B(K)=0:A=K:GOSUB 760
DD 190 A=T:GOTO 760
# 200 GOSUB 520: IF S=1 THEN PRINT"The witches wi
      n!":GOTO 22Ø
M 210 PRINT"The ghosts win!"
OF 220 LOCATE 23,10:PRINT"Hit a key to play again
OE 230 K$=INKEY$:IF K$="" THEN 230
GE 240 RUN
NI 25Ø S=-S:H=Ø:A=7
LP 260 IF A=36 THEN 200
ML 270 GOSUB 60:IF H=0 THEN A=A+1:GOTO 260
66 280 D=0:GOSUB 520:IF S=1 THEN PRINT"Ghost's tu
```

```
OF 290 PRINT"Witch's turn"
8M 3ØØ PRINT TAB(16) "From: ";
N 310 E=E+1:K$=INKEY$:IF K$="" THEN 310
88 320 IF ASC(K$)=13 THEN GOSUB 520: RANDOMIZE E:G
      OTO 20
60 33Ø IF ASC(K$)<97 OR ASC(K$)>121 THEN 31Ø
DM 340 PRINT K$: A=N(ASC(K$)-97): Z=A
DB 350 LOCATE 23,18:PRINT"To:";
60 360 K$=INKEY$: IF K$="" THEN 360
DD 370 IF ASC(K$)<97 OR ASC(K$)>121 THEN 360
JN 380 PRINT K$: T1=N(ASC(K$)-97)
50 390 CK=1:L=0:K1=0:GOSUB 60:CK=0
KE 400 H=0:A=7
BH 410 IF A=36 THEN 440
ME 42Ø GOSUB 6Ø: IF H>=1 THEN 44Ø
FH 43Ø A=A+1: IF A<36 THEN 42Ø
0L 44Ø IF D THEN 47Ø
DN 45Ø IF L THEN 48Ø
6F 46Ø SOUND 99,5:GOTO 28Ø
FO 470 IF L=0 OR K1=0 THEN SOUND 99,5:50TO 510
09 48Ø IF K1=Ø AND H>=1 THEN 46Ø
CD 490 F=Z:T=T1:K=K1:GOSUB 170:IF K1=0 THEN 250
AG 500 A=T: 7=A: H=0: GOSUB A0: IF H<1 THEN 250
JD 510 GOSUB 520: D=1:GOTO 350
MO 520 LOCATE 20.1: FOR B=1 TO 3: PRINT: PRINT"
                                 "::NEXT:LOCATE 21
       14: RETURN
AE 53Ø KEY OFF: SCREEN 1: COLOR Ø, 1: CLS: DEFINT C, W
LI 540 DIM C1(98), W1(98), SQ(98), B(42), D(28), X(35)
      ,Y(35),L(35),XL(35),YL(35),N(28)
LP 550 LINE (50,80)-(81,103),1,B
IH 560 LOCATE 12,12: PRINT "The Witching Hour
DA 570 LINE (230,80)-(261,103),1,B
NN 580 GET (50,80)-(81,103),50
II 590 FOR A=0 TO 52: READ C1(A): NEXT
@M 600 PUT (56,82),C1
MA 610 GET (50,80)-(81,103),C1
NC 620 FOR A=0 TO 69: READ W1(A): NEXT
00 63Ø PUT (232,82),W1
MN 64Ø GET (23Ø,9Ø)-(261,1Ø3),W1
II 650 S=-1:FOR A=0 TO 7:READ M(A):NEXT
PN 660 FOR A=0 TO 28: READ D(A): NEXT
LD 67Ø B=48:C=32:D=59:E=12
QL 680 FOR A=0 TO 4:FOR F=0 TO 4:H=6*A+F+7:X(H)=B
      *F+D-15: Y(H) =C*A+E-11
BO 690 L(H)=G+97:N(G)=H:G=G+1:XL(H)=6*F+10:YL(H)=
      4*A+1:NEXT:NEXT
AF 700 CLS:FOR A=0 TO 4:LINE (D,C*A+E)-(B*4+D,C*A
```

+E),2:NEXT

```
BH 710 FOR A=0 TO 4:LINE (B*A+D,E)-(B*A+D,C*4+E),
      2: NEXT
OE 720 A=0:F=0:GOSUB 740:A=B+B:GOSUB 740:F=C+C:GO
      SUB 740: A=0: GOSUB 740
EF 730 FOR A=0 TO 42: READ B(A): GOSUB 760: NEXT: RET
      URN
FP 740 LINE (D+A,E+F)-(2*B+D+A,2*C+E+F),2
FJ 750 LINE (D+A, 2*C+E+F) - (2*B+D+A, E+F), 2: RETURN
KG 760 IF B(A)=2 THEN RETURN
FP 770 IF B(A) (0 THEN PUT (X(A), Y(A)), W1, PSET
H# 780 IF B(A)=0 THEN PUT (X(A),Y(A)),SQ,PSET
KL 790 IF B(A)>0 THEN PUT (X(A), Y(A)), C1, PSET
JH 800 LOCATE YL(A), XL(A):PRINT CHR$(L(A)):RETURN
N 810 DATA 36,20,-256,192,0,-961,0,16128,255,0,-
      1,192,-3329,-16177,Ø,-1,192
th 820 DATA 16128,255,0,-1009,-16381,1020,16368,-
      16,-1,-3841,-1,-769,-16336,-193
FC 83Ø DATA 192,16128,-3841,Ø,-241,252,768,-769,Ø
      , 16128, 252, Ø, -4Ø33, Ø, 16128, 192, Ø
N 840 DATA -4081,0,0,255,0
@1 850 DATA 52,20,48,0,3,15360,240,768,0,-241,252
      ,-16381,768,-1,960,192,-256
CP 860 DATA -769, -4081, 0, -241, 16383, 255, 3840, 4095
      ,12543,0,-1009,-1,192,768,-61
№ 87Ø DATA 207, Ø, 384Ø, -16129, Ø, Ø, -12289, 192, Ø, -2
      53,-16369,0,768,-15361,240
$P 880 DATA -32768, -253, 15600, 0, 168, -193, -16369, -
      22016,-81,-21761,-24406,16296
HL 890 DATA -1,0,-32768,-241,252,0,3840,-3841,0,0
      -16372
ML 900 DATA -6,1,6,-1,-5,7,5,-7
MN 910 DATA 7,3,7,3,7,0,3,7,3,7,3,0
DN 92Ø DATA 7,3,7,3,7,Ø,3,7,3,7,3,Ø,7,3,7,3,7
II 930 DATA 2,2,2,2,2,2,-1,-1,-1,-1,-1,2
```

CI 940 DATA -1,-1,-1,-1,-1,2,-1,-1,0,1,1,2 BJ 950 DATA 1,1,1,1,1,2,1,1,1,1,2,2,2,2,2,2,2,2

# **Jackpot**

### Rick Rothstein IBM PC/PCir version by Kevin Mykytyn

Now you can experience the thrill of slot machines without the danger of losing your money. These programs will show you how the bandits work and also how difficult it really is to hit a jackpot. For the IBM PC (color/graphics adapter required) and PCir.

Have you ever been to a casino in Las Vegas or Atlantic City? If so, your first visit probably left you dumbstruck over the sheer number of slot machines waiting to take your money. These nefarious one-arm bandits dazzle you with bright lights

and promises of instant wealth.

A recent trip to Atlantic City—and an unprofitable encounter with some of these machines—prompted me to write "Jackpot." This IBM program runs on both the IBM PC (as long as you have a color/graphics adapter) under BASICA, and the IBM PCjr using Cartridge BASIC. There's one level of play—the jackpot payout is random and follows no casino-like odds. You win some, you lose some.

# **Bells and Whistles**

Colorful graphics are used to display a payout chart, your current monetary status, and three large windows through which cherries, limes, apples, carrots, bars, bananas, or lucky sevens will show. The shape displayed in each window is picked at random from ten-position wheels which hold a random scattering of all seven shapes.

Payout?

Jackpot is one of the simplest games you'll find. Just press the P key and the wheels spin, clack a bit, and come to rest. Lost another? Don't worry. All you've lost is another point from your starting kitty of 100. Won one? A delightful sound reaches your ears and your total goes up appropriately.

When your arm is tired from pulling the electronic lever, give the bandit a rest by pressing the E key. What could be

easier?

Easy to quit? Not really. Gambling is addicting, and though your money's not at risk here, you'll still thrill at the thought of three sevens, and feel the anguish of defeat when the jackpot spins and spins, and rewards you with a bunch of bananas.

### Jackpot

- NH 10 DEFINT A-Z:SCREEN 1:KEY OFF
- IK 20 DEF SEG-0: POKE 1047,64
- PM 3Ø GOSUB 43Ø:DIM A(E):A(Ø)=X:A(1)=Y:FOR I=2 TO E:READ A(I):NEXT
- HI 4Ø GOSUB 43Ø:DIM L(E):L(Ø)=X:L(1)=Y:FOR I=2 TO E:READ L(I):NEXT
- 0M 60 GOSUB 430:DIM S(E):S(0)=X:S(1)=Y:FOR I=2 TO E:READ S(1):NEXT
- OA 70 GOSUB 430:DIM CA(E):CA(0)=X:CA(1)=Y:FOR I=2
  TO E:READ CA(I):NEXT
- KJ 8Ø GOSUB 43Ø:DIM CH(E):CH(Ø)=X:CH(1)=Y:FOR I=2 TO E:READ CH(I):NEXT
- E6 9Ø GOSUB 43Ø:DIM BN(E):BN(Ø)=X:BN(1)=Y:FOR I=2
  TO E:READ BN(I):NEXT
- NC 100 COLOR 0,2:CLS:GOSUB 1300:T=100
- EE 110 LINE (76,25)-(112,60),2,B:LINE (142,25)-(1 78,60),2,B:LINE (208,25)-(244,60),2,B
- KL 12Ø PUT (10,75),S:PUT (50,75),S:PUT (90,75),S: PUT (200,75),S:PUT (240,75),S
- 16 13Ø PUT (10,100), CH:PUT (50,100), CH:PUT (90,100), CH:PUT (200,100), CH:PUT (240,100), CH
- AD 140 PUT (10,125),L:PUT (50,125),L:PUT (90,125),L:PUT (200,125),CH
- 8K 15Ø PUT (10,150),A:PUT (50,150),A:PUT (90,150)
  ,A:PUT (200,150),A:PUT (240,150),A
- PO 160 PUT (10,175), CA:PUT (50,175), CA:PUT (90,175), CA:PUT (200,175), CA:PUT (240,175), CA
- IL 170 LÓCATE 12,17:PRINT " 25":LOCATÉ 12,36:PRIN T" 10"
- 60 180 LOCATE 15,17:PRINT " 15":LOCATE 15,34:PRIN T " 5"
- 0K 190 LOCATE 18,17:PRINT " 10":LOCATE 18,36:PRIN T " 2"
- HI 200 LOCATE 21,17:PRINT " 18":LOCATE 21,36:PRIN T " 10"
- AB 210 LOCATE 24,17:PRINT " 14":LOCATE 24,36:PRIN T " 10";:GOSUB 420
- # 22Ø LOCATE 8,5:PRINT "Press (P) to play or (E)
  to end";

- D 230 IF T<=0 THEN LOCATE 7,5:PRINT "Sorry, you are broke. Play again ? (y/n)":60TO 440</p>
- EC 24Ø H=Ø:W=Ø:A\$=INKEY\$:A=RND(1):IF A\$<>"E" AND A\$<>"P" THEN 24Ø
- CC 250 IF A\$="E" THEN CLS:END
- 6F 26Ø X=79:Y=24:WH=1:GOSUB 32Ø:X=145:WH=2:GOSUB
  32Ø:X=211:WH=3:GOSUB 32Ø
- LJ 270 FOR A=1 TO 24 STEP 2:H\$=STR\$(H):H\$=RIGHT\$(
  H\$,(LEN(H\$)-1))
- !P 28Ø L=LEN(P\$(A)):IF P\$(A)=LEFT\$(H\$,L) THEN W=V
  AL(P\$(A+1))
- IN 290 NEXT: IF W>0 THEN GOSUB 400
- HK 300 T=T-1:GOSUB 420
- 00 310 POKE 1050, PEEK (1052): GOTO 230
- LA.320 FOR J=1 TO RND(1)\*6+5:K=INT(RND(1)\*17)+1:0
  N G(WH,K) GOSUB 330,340,350,360,370,380,39
  0:SOUND 20\*K+37,.1:FOR TD= 1 TO J\*40:NEXT:
  NEXT:H=H\*10+G(WH,K):RETURN
- BH 330 PUT (X,Y),B,PSET:RETURN
- CI 340 PUT (X,Y),S,PSET:RETURN
- BM 350 PUT (X,Y),A,PSET:RETURN
- M 360 PUT (X,Y),CH,PSET:RETURN
- LF 370 PUT (X,Y),L,PSET:RETURN
- MC 380 PUT (X,Y), CA, PSET: RETURN
- IF 390 PUT (X,Y), BN, PSET: RETURN
- HH 400 IF W=101 THEN PLAY SONG\$:T=T+1:FOR A=1 TO 25:T=T+4:GOSUB 420:NEXT:RETURN
- NK 410 FOR A=1 TO W:T=T+1:GOSUB 420:FOR B=1531 TO 1540:SOUND B, 1:NEXT:NEXT:RETURN
- 80 420 LOCATE 1,5:PRINT "Winnings "T-100" ":LOCATE 1,25:PRINT "Total "T" ":RETURN
- KI 430 READ X, Y: E= (4+INT ((X+7)/8) \*Y)/2: RETURN
- JF 440 AS=INKEYS: IF AS<>"Y" AND AS<>"N" THEN 440
- DB 450 IF A\$="Y" THEN LOCATE 7,5:PRINT "

  ONE MOMENT PLEASE ":RUN ELSE
  CLS:END
- LC 460 DATA &H40,&H17,&H0,&H1400,&H0,&H0,&H0,&H50
- ወ ዙ 470 Data &Hø,&Hø,&Hø,&H1øø,&H4ø,&Hø,&Hø,&HA9øA
- LF 480 DATA &HA06A,&H0,&H0,&HAA2A,&HA8AA,&H0,&H0, &HAAAA
- FC 490 DATA &HAAAA,&HØ,&H200,&HAAAA,&HAAAA,&H80,& HA00,&HAAAA
- 6P 500 DATA &HAAAA,&HA0,&H2A00,&HAAAA,&HAAAA,&HAB ,&H2A00,&HAAAA
- FF 510 DATA &HAAAA, &HAB, &HAAØØ, &HAAAA, &HAAAA, &HAA , &HAAØØ, &HAAAA
- LL 52Ø DATA &HAAAA, &HAA, &HAAØØ, &HAAAA, &HAAAA, &HAAAA, &HAAAA

- MP 53Ø DATA &HAAAA, &HAA, &HAAØØ, &HAAAA, &HAAAA, &HAA , &H2AØØ, &HAAAA
- AE 540 DATA &HAAAA, &HA8, &H2A00, &HAAAA, &HAAAA, &HA8, &HA00, &HAAAA
- CE 55Ø DATA &HAAAA,&HAØ,&H2ØØ,&HAAAA,&HAAAA,&H8Ø, &HØ,&HAAAA
- MG 56Ø DATA &HAAAA,&HØ,&HØ,&HAA2A,&HABAA,&HØ,&HØ, &HAAØA
- BK 570 DATA &HAOAA, &HO, &HO, &HA800, &HZA, &HO, &HO
- LI 580 DATA &H40, &H17, &H0, &H0, &H0, &H0, &H0, &H0
- EJ 590 DATA &HØ,&HØ,&HØ,&HØ,&HØ,&HØ,&HØ,&H5501
- KB 6ØØ DATA &H4Ø55,&HØ,&HØ,&H5515,&H5455,&HØ,&H1Ø Ø,&H5555
- IF 610 DATA &H5555, &H40, &H1500, &H5555, &H5555, &H54, &H5500, &H5555
- P6 62Ø DATA &H5555, &H55, &H55Ø1, &H5555, &H5555, &H4Ø 55, &H55Ø5, &H5555
- € 63Ø DATA %H5555, %H5Ø55, %H5515, %H5555, %H5555, %H 5455, %H5555, %H5555
- 6A 64Ø DATA &H5555,&H5555,&H5515,&H5555,&H5555,&H 5455,&H55Ø5,&H5555
- 60 65Ø DATA &H5555,&H5Ø55,&H55Ø1,&H5555,&H5555,&H 4Ø55,&H55ØØ,&H5555
- 08 66Ø DATA &H5555, &H55, &H15ØØ, &H5555, &H5555, &H54, &H1ØØ, &H5555
- 6K 67Ø DATA &H5555,&H4Ø,&HØ,&H5515,&H5455,&HØ,&HØ,&HØ,&HS501
- BL 68Ø DATA &H4Ø55, &HØ, &HØ, &HØ, &HØ, &HØ, &HØ, &HØ
- €€ 69Ø DATA &HØ,&HØ,&HØ,&HØ,&HØ,&HØ,&HØ
- IE 700 DATA %H40,%H17,%HAAAA,%HAAAA,%HAAAA,%HAAAA,%HAAAA
- 9A 71Ø DATA &HAAAA, &HAAAA, &HAZAA, &HAAA&, &H8AZA, &H AAAZ, &H8ØAA, &HZ8AØ
- F6 72Ø DATA &H2ØA,&HAA8Ø,&HAØ,&HØ,&HØ,&HAØØ,&HAAA 2,&H2A8Ø
- #I 73Ø DATA &HAA8,&HAAA,&HAAA2,&HAAAØ,&HAAA,&H8AA A,&H8ØA2,&HAØAØ
- NL 740 DATA &HAØA,&H8AØ2,&H8ØA2,&HAØAØ,&HAØA,&H8A Ø2,&H8ØA2,&HAØAØ
- IA 750 DATA &HAØA,&HBAØ2,&HAAA2,&HAABØ,&HAAA,&HAAAA,&HAAAA,&HAAABØ
- EL 760 DATA &HAAA, &HBAAA, &HBØA2, &HAØAØ, &HAØA, &HBA Ø2, &HBØA2, &HAØAØ
- NB 770 DATA &HAØA, &HBAØ2, &HBØA2, &HAØAØ, &HAØA, &HBA
  Ø2, &HBØA2, &HAØAØ
- EL 78Ø DATA &HAØA, &HBAØ2, &HB2A2, &HAØAØ, &HAØA, &HBA Ø2, &HAAA2, &HAØBØ
- EB 790 DATA &HAØA,&H8AØ2,&HAØ,&HØ,&HØ,&HAØØ,&H8ØA A,&H28AØ

- B 800 DATA &H20A,&HAAB0,&HA2AA,&HAAAB,&HBA2A,&HAAAAA,&HAAAA
- MC 810 DATA &HAAAA,&HAAAA,&HAAAA,&HAAAA,&HAAAA,&H
- M 920 DATA &H40,&H17,&H0,&HAAAA,&HA8AA,&H0,&H0,&H0,&H0
- CM 83Ø DATA &HABAA, &HØ, &HØ, &HØ, &HABØØ, &HØ, &HØ, &HØ
- № 84Ø DATA &HAØØ2, &HØ, &HØ, &HØ, &HBØØA, &HØ, &HØ, &HØ
- E0 850 DATA &H2A, &H0, &H0, &H0, &HAB, &H0, &H0, &H200
- 8N 86Ø DATA &HAØ,&HØ,&HØ,&HAØØ,&HBØ,&HØ,&HØ,&H2AØ Ø
- NH 870 DATA &HØ, &HØ, &HØ, &HA800, &HØ, &HØ, &HØ, &HA002
- EH 880 DATA &HO, &HO, &HO, &HADOZ, &HO, &HO, &HO, &HADOZ
- N 890 DATA &HØ, &HØ, &HØ, &HAØØ2, &HØ, &HØ, &HØ, &HAØØ2
- EI 900 DATA &HO, &HO, &HO, &HA002, &HO, &HO, &HO, &HA002
- EK 710 DATA &HØ, &HØ, &HØ, &HAØØ2, &HØ, &HØ, &HØ, &HAØØ2
- EM 920 DATA &HØ, &HØ, &HØ, &HAØ02, &HØ, &HØ, &HØ, &HAØ02
- 6J 93Ø DATA &HØ, &HØ, &HØ, &HØ, &HØ, &HØ, &HØ
- KE 940 DATA &H40,&H17,&H0,&H0,&H0,&H0,&H0,&H0
- NF 950 DATA &HØ, &HØ, &HØ, &HØ, &HØ, &HØ, &HØ
- NH 960 DATA &HØ, &HØ, &HØ, &HØ, &HØ, &HØ, &HØ, &HØ
- 01 970 DATA &HØ, &HØ, &HØ, &HØ, &HØ, &HØ, &HØ, &HØ
- NH 98Ø DATA &HØ,&H1ØØ,&HØ,&HFFØØ,&HFFFF,&H1FC,&HØ,&HFFFF
- DK 990 DATA &HFFFF, &H4FF, &HFFØØ, &HFFFF, &HFFFF, &HD
- 60 1000 DATA &HFFFF, &HD4FF, &HFFØ0, &HFFFF, &HFFFF, &HD5FF, &HØ, &HFFFF
- DE 1010 DATA &HFFFF, &H4FF, &H0, &HFF00, &HFFFF, &H1FC , &H0, &H0
- 14 1020 DATA &HØ, &H100, &HØ, &HØ, &HØ, &HØ, &HØ, &HØ
- 8K 1Ø3Ø DATA &HØ, &HØ, &HØ, &HØ, &HØ, &HØ, &HØ, &HØ
- BN 1040 DATA &HO,&HO,&HO,&HO,&HO,&HO,&HO,&HO
- NB 1050 DATA &HØ, &HØ, &HØ, &HØ, &HØ, &HØ, &HØ
- FA 1060 DATA &H40, &H17, &H0, &H0, &H0, &H0, &H0, &H0
- BS 1070 DATA &HØ, &HØ, &HØ, &HØ, &HØ, &HØ, &HØ, &HØ
- CJ 1080 DATA &H40, &H0, &H0, &H0, &H40, &H0, &H0, &H0
- AC 10990 DATA &H40, &H0, &H0, &H100, &H0, &H0, &H0, &H40
  0
- ଧ 1110 DATA &H400,&H0,&H0,&H1,&H2A00,&H80,&H0,&H 802A
- 1120 DATA &HAA00, &H30, &H0, &HA0AA, &HAB02, &HE8, & H200, &HA8FA
- JM 1130 DATA &HAA02,&HA8,&H200,&HA8EA,&HAA02,&HA8, &H200,&HA8AA
- LD 1140 DATA &HAA02,&HA8,&H200,&HA8AA,&HAA00,&HA0,&HA0

- J6 115Ø DATA &H2AØØ, &H8Ø, &HØ, &H8Ø2A, &HØ, &HØ, &HØ, & HØ
- BF 1160 DATA &HØ, &HØ, &HØ, &HØ, &HØ, &HØ, &HØ, &HØ
- NG 1170 DATA &HØ, &HØ, &HØ, &HØ, &HØ, &HØ, &HØ
- UJ 1180 DATA &H40, &H17, &H0, &H0, &H0, &H0, &HC000
- CD 1190 DATA &HØ, &HØ, &HØ, &H3003, &HØ, &HØ, &HØ, &HCOC
- N 1200 DATA &HØ, &HØ, &HØ, &HF30, &HCØ, &HØ, &HØ, &HF3C
- CP 1210 DATA &HFC, &HØ, &HØ, &H33F, &HFF, &HØ, &HØ, &HC3 3F
- LD 1220 DATA &HCOFF, &HO, &HO, &HFO3F, &HFOFF, &HO, &HO . &HFCØF
- IC 1230 DATA &HFC3F, &HØ, &HØ, &HFFØF, &HFFØF, &HØ, &HØ
- EE 1240 DATA &HFFC3,&HC0,&H0,&HFF03,&HFFF0,&HF0,& HØ, &HFFØØ
- 6L 125Ø DATA &HFFC, &HFC, &HØ, &H3FØØ, &HFF, &HFF, &HØ, &HFØØ
- 1260 DATA &HCOFF, &HCOO3, &HO, &H300, &HFOFF, &HO, & HØ, &HØ
- MK 1270 DATA &HFC3F, &HØ, &HØ, &HØ, &HFFØ3, &HØ, &HØ, &H Ø
- KF 128Ø DATA &H3ØØ, &HCØ, &HØ, &HØ, &HØ, &HØ, &HØ, &HØ
- 00 129Ø DATA &HØ, &HØ, &HØ, &HØ, &HØ, &HØ, &HØ
- PK 1300 DIM G(3,17):FOR A=1 TO 3:FOR B=1 TO 17:RE AD G(A, B): NEXT: NEXT
- ## 1310 DATA 1,2,1,4,5,6,7,5,7,3,5,5,6,7,3,2,5
- IL 1320 DATA 1,2,3,7,5,6,7,5,7,3,4,5,6,7,3,2,3
- KJ 133Ø DATA 1,2,3,4,5,6,7,5,7,3,4,5,6,7,5,2,3 NI 134Ø DIM P\$(24):FOR A=1 TO 24:READ P\$(A):NEXT
- LP 1350 DATA 4.3,44,6,444,16,555,11,66,11,666,15, 33, 11, 333, 19, 22, 11, 222, 26, 11, 26, 111, 101
- LE 1360 SONG\$="mb t150 o3 18 eg. 116 e 18 fg...eg .116e18fg...g o4ecdccdecdc ":RETURN

# **Balloon Crazy**

Joseph Russ IBM PC/PCjr version by Charles Brannon

Catch as many balloons as you can—but be careful not to fall off your skateboard. This whimsical game runs on the IBM PC (with color/graphics adapter and BASICA) and IBM PCjr (with Cartridge BASIC). Requires a joystick.

"Balloon Crazy" is a game that children can enjoy, yet its higher levels are a challenge for adults. The goal is simple—you must zip back and forth across the screen while catching falling balloons on top of your head. Since some of the balloons fall very fast, that's not as easy as it sounds. After you've caught enough balloons (six in most versions), you can reach up to pop them, then catch some more. If you miss just one, you lose all the balloons currently in your possession.

Type in Balloon Crazy from the listing below, then save a

copy of it before you try to run it.

# **Oodles of Balloons**

You'll need a joystick to play Balloon Crazy on the IBM PC or PCjr. You may want to unlock the horizontal axis of the joystick. Before the game begins, you have an opportunity to adjust the joystick if needed. Press Y when prompted and follow the instructions on the screen.

Balloon Crazy begins by displaying several rows of multicolored balloons at the top of the screen. You're the tiny clownlike figure at the bottom. When a balloon begins to fall, move directly under it and catch it on your head. As soon as you snare a balloon, it joins the pile on top of your head.

Miss a balloon, and you tumble to the ground. All the balloons on your head fall and pop. In a flash, you bounce away, then, defying gravity, leap up and off the top of the

screen.

Once you've stacked the required number of balloons on top of your head (the number depends on how many rows of balloons are left on the screen—at first, only four are needed), you can pop them all. By the time you're clearing out the topmost level of balloons, you have to hold seven of them in a delicate balance. Clear a screen and you're advanced to the next level.

You score ten points for every balloon you catch and pop. With three players—you lose one each time you let a balloon get by you—try to catch screenful after screenful of bright balloons.

### **Crazed Change**

You can adjust the difficulty of Balloon Crazy by changing the statement DF=10 in line 120. The variable DF controls how close you must be to a balloon to catch it. Changing DF to a higher value makes the game easier, and decreasing it makes the game more difficult.

### **Balloon Crazy**

- KL 100 'Balloon Crazy for IBM PC/PCjr requires BA SICA, Color Graphics adapter, and one joys tick
- 0K 110 DEFINT A-Z:RANDOMIZE TIMER:DIM BP\$(3),MOBJ
  (452),X(49),Y(49)
- P 120 DF=10:' change df to a smaller number for a greater challenge
- 10 13Ø SCREEN 1:COLOR 9,Ø:KEY OFF:STRIG ON::PLAY
   "mf":CLS
- AD 140 GOSUB 280: X=100:HF=164:EY=Y+22
- 16 150 TX=3:LX=134:SKEW!=2.27:SP!=4:LIVES=4
- E 160 GOSUB 740:PRINT"BALLOON C R A Z Y !":LOCAT E 9,10:PRINT"/":LOCATE 8,11:PRINT"Do you n eed":LOCATE 9,11:PRINT"to adjust":LOCATE 1 0,11:PRINT"your joystick?":WHILE INKEY\$<>"".WEND
- El 170 A\$=INKEY\$:IF A\$="" AND STRIG(1)=0 THEN 170 ELSE IF (ASC(A\$+CHR\$(0)) OR 32)=121 THEN GOSUB 600
- KE 180 CLS:PRINT"BALLOON C R A Z Y !":LOCATE 1,25
  :PRINT"Score:";SC!
- MI 190 FOR I=158 TO 158 +(LIVES-2)\*8 STEP 8:PUT(I,0),TINY:NEXT:GOSUB 350
- KE 200 FOR ROW=20 TO 80 STEP 20:FOR COL=45 TO 255 STEP 15:PUT (COL,ROW), BALL:NEXT COL,ROW
- QJ 21Ø PUT(X,Y), MAN: GET (X,Y)-(X+21,EY), MOBJ

- J6 23Ø PUT(BX,BY),BALL:BY=BY+SP!:FUT(BX,BY),BALL:
   IF BY>18Ø THEN 44Ø
- K! 24Ø IF ABS(BY-HP)<SP! THEN IF ABS((BX-7)-X)<DF
   THEN Y=Y-13:PUT(BX,BY),BALL:PUT(X+7,Y),BA
   LL,PSET:SOUND 30000,1:GET(X,Y)-(X+21,EY),M
   OBJ:HP=HP-13:FLOATERS=FLOATERS+1:SP!=SP!+.
  5:IF FLOATERS=7-BP THEN GOSUB 380:GOTO 210
   ELSE 220</pre>
- FD 25Ø PUT(X,Y),MOBJ:NX=(STICK(Ø)-TX)\*SKEW::X=NX:
  IF NX<Ø THEN X=Ø ELSE IF NX>297 THEN X=297
- HE 260 PUT (X,Y), MOBJ
- 06 27Ø GOTO 23Ø
- PD 280 READ XS,YS:E=(4+INT((XS+7)/8)\*YS)/2:DIM MA N(E):MAN(Ø)=XS:MAN(1)=YS:FOR I=2 TO E:READ MAN(I):NEXT:Y=200-YS:MY=Y
- # 290 READ XS,YS:E=(4+INT((XS+7)/8)\*YS)/2:DIM PO P(E):POP(0)=XS:POP(1)=YS:FOR I=2 TO E:READ POP(I):NEXT
- HO 300 READ XS,YS:E=(4+INT((XS+7)/8)\*YS)/2:DIM FA
  LL(E):FALL(0)=XS:FALL(1)=YS:FOR I=2 TO E:R
  EAD FALL(I):NEXT
- #E 310 READ XS,YS:E=(4+INT((XS+7)/8)\*YS)/2:DIM BA LL(E):BALL(0)=XS:BALL(1)=YS:FOR I=2 TO E:R EAD BALL(I):NEXT
- KE 32Ø READ XS,YS:E=(4+INT((XS+7)/8)\*YS)/2:DIM XB
  ALL(E):XBALL(Ø)=XS:XBALL(1)=YS:FOR I=2 TO
  E:READ XBALL(I):NEXT
- CL 33Ø READ XS,YS:E=(4+INT((XS+7)/8)\*YS)/2:DIM TI
  NY(E):TINY(Ø)=XS:TINY(1)=YS:FOR I=2 TO E:R
  EAD TINY(I):NEXT
- MF 340 RETURN
- LG 350 BP=3:FOR I=0 TO BP:BP\$(I)="ABCDEFGHIJKLMNO":NEXT
- NJ 360 RETURN
- JH 37Ø GOSUB 38Ø:GOTO 22Ø
- LE 380 FOR I=1 TO FLOATERS
- LK 39Ø PUT(X,MY),POP,PSET:PUT(X+7,MY-13),BALL:PUT
  (X+7,MY-18),XBALL:FOR J=Ø TO 5:SOUND 100+J
  ,.5:NEXT:PUT(X+7,MY-18),XBALL
- 00 400 IF I<FLOATERS THEN PUT(X+7,Y),BALL:PUT(X+7,MY-13),BALL</pre>
- FL 410 PUT(X,MY), MAN, PSET: FOR W=1 TO 5: NEXT: SC!=S C!+10:LOCATE 1,31: PRINT SC!: Y=Y+13: NEXT
- 0H 42Ø Y=MY:GET (X,Y)-(X+21,EY),MOBJ:PUT (X,Y),MA

- PO 430 SP!=4:HP=164:FLOATERS=0:RETURN
- PD 440 PUT(BX,BY),BALL:PUT(BX,BY-5),XBALL:FOR J=0
  TO 5:SOUND 105-J,.5:NEXT:PUT(BX,BY-5),XBA
- EH 450 PUT (X, MY), MAN: PUT (X, MY), FALL
- CC 460 IF FLOATERS=0 THEN 510
- LD 470 FOR I=1 TO FLOATERS
- 01 480 PUT(X+7,MY-13),BALL:PUT(X+7,MY-18),XBALL:F OR J=0 TO 5:SOUND 105-J,.5:NEXT:PUT(X+7,MY-18),XBALL
- PF 49Ø IF I<FLOATERS THEN PUT(X+7,Y),BALL:PUT(X+7
  ,MY-13),BALL</pre>
- JM 500 FOR W=1 TO 5:NEXT: Y=Y+13:NEXT
- FL 51Ø NY=MY:S=-6:FOR I=X+5 TO 291 STEP 5:PUT(I-5
  ,NY),FALL:NY=NY+S:IF NY<MY-18 OR NY>MY THE
  N NY=NY-S:S=-S
- ₩ 52Ø PUT(1,NY), FALL: SOUND 100+NY, .5: NEXT
- HO 530 NX=I-5:FOR I=NY-16 TO 0 STEP-16:PUT(NX,I+1 6),FALL:PUT(NX,I),FALL:SOUND 5000-I\*5,.1:N EXT
- 1 540 PUT (NX, I+16), FALL
- JM 550 LIVES=LIVES-1:PUT(150+LIVES\*8,0),TINY:IF L IVES>0 THEN PUT(X,MY),MAN:GOSUB 420:GOTO 2 10
- OM 56Ø CLS:FOR I=Ø TO 49:X(I)=4+15\*INT(2Ø\*RND(1))
  :Y(I)=18\*INT(1Ø\*RND(1)):PUT (X(I),Y(I)),BA
  LL.PSET:NEXT
- JA 570 FOR I=0 TO 49:PUT (X(I)-4,Y(I)),XBALL,PSET :SOUND 100+5\*RND(1),.2:SOUND 30000,.2:PUT (X(I)-4,Y(I)),XBALL:NEXT
- LL 580 LOCATE 12,16:PRINT"GAME OVER":LOCATE 13,17
  -LEN(STR\$(SC!))/2:PRINT"Score:";SC!:LOCATE
  14,15:PRINT"Press Button"
- N 590 A\$=INKEY\$:IF INKEY\$="" AND STRIG(1)=0 THEN 590 ELSE RUN
- KP 600 GOSUB 740
- JN 610 LOCATE 9,10:PRINT"/":LOCATE 8,11:PRINT"Mov
   e stick to":LOCATE 9,11:PRINT"far left,":L
   OCATE 10,11:PRINT"press button!"
- JI 420 TX=STICK(0): IF STRIG(1)<>0 THEN 640
- JI 630 FOR I=10 TO 0 STEP-1:C=-C\*(C<3)+1:LINE (30+1,40)-(1,70),C:LINE-(30+1,100),C:NEXT:GOT 0 620
- W 640 PUT(50,63), POP, PSET: PUT(57,45), XBALL, PSET: FOR J=1 TO 15: SOUND 100+J,.5: NEXT
- B6 65Ø CLS:PUT (5Ø,63),MAN:LOCATE 9,10:PRINT"/":L OCATE 8,11:PRINT"Gimme another":LOCATE 9,1 1:PRINT"balloon!"
- CJ 660 FOR J=1 TO 2000: NEXT

- KM 670 PUT(57,0),BALL:FOR I=2 TO 50 STEP 2:WAIT &
  H3DA,8:PUT (57,I-2),BALL:WAIT &H3DA,8:PUT
  (57,I),BALL:NEXT
- 06 68Ø LOCATE 8,11:PRINT"Move stick to ":LOCATE 9
   ,11:PRINT"far right,":LOCATE 1Ø,11:PRINT"p
   ress button!"
- CB 69Ø LX=STICK(Ø): IF STRIG(1)<>Ø THEN 71Ø
- P 700 FOR I=10 TO 0 STEP-1:C=-C\*(C<3)+1:LINE (28
  9-I,40)-(319-I,70),C:LINE-(289-I,100),C:NE
  XT:GOTO 690</pre>
- MK 710 FOR I=1 TO 5:PUT (57,45),XBALL,PSET:PUT(50,63),POP,PSET:FOR J=1 TO 4:SOUND 100+J,.5:
   NEXT:PUT(57,50),BALL,PSET:PUT(50,63),MAN,P
   SET:FOR J=1 TO 100:NEXT:NEXT
- NA 720 SKEW!=297/ABS(LX-TX)
- MH 73Ø RETURN
- PC 740 CLS:PUT(0,63),MAN:PUT (57,0),BALL:FOR I=2 TO 50 STEP 2:PUT (57,I-2),BALL:PUT (57,I), BALL:PUT (I-2,63),MAN:PUT (I,63),MAN,PSET: NEXT:RETURN
- HA 75Ø DATA &H2C, &H17, &HØ, &H5, &HØ, &HØ, &H4Ø15, &HØ
- FD 760 DATA &HØ,&H5055,&HØ,&HØ,&H30CF,&HØ,&H300,& HCC3
- J 770 DATA &HØ, &H300, &HACAA, &HØ, &HØ, &HAØAØ, &HØ, & HØ
- P 78Ø DATA &H8Ø2A,&HØ,&HØ,&HF,&HØ,&HEØØ,&HEEEE,& HCØ
- FA 790 DATA &HFBØØ, &HBBBB, &HBØ, &HCØØ3, &HEØEE, &H3C , &HF, &HBØ3B
- M 800 DATA &HF,&HFF,&HC02E,&HF00F,&H0,&H4015,&H0
- 6J 810 DATA &HAØAA, &HØ, &HØ, &HAØAA, &HØ, &H2ØØ, &HABA Ø, &HØ
- FJ 820 DATA &H200, &HABA0, &H0, &HA00, &H2AB0, &H0, &HA 37, &H2AB0
- AB 83Ø DATA &HCØØD,&HDADD,&H3ACØ,&H7Ø77,&H7737,&H 1D4Ø,&HCØDD,&HA8ØØ
- 6P 84Ø DATA &H2C,&H17,&H8ØØ,&H5,&HØ,&HAØØØ,&H4Ø15,&HØ
- IG 850 DATA &H8003, &H5055, &H0, &HC003, &HC3, &H0, &H3 0F, &H3CCF
- KH 860 DATA &HØ,&H30F,&HACAA,&HØ,&HC003,&HA0A0,&H0,&HF003
- KI 870 DATA &H802A, &H0, &HFC00, &HF, &H0, &H2E00, &HEE EE, &HC0
- N 880 DATA &HB00, &HBBBB, &HBC, &H0, &HE0EE, &HFF, &H0, &H803B
- EG 890 DATA &HF,&H0,&HC02E,&HC003,&H0,&H4015,&HC0

- 60 900 DATA &HAØAA, &HCØØØ, &HØ, &HAØAA, &HØ, &H2ØØ, &H ABAØ, &HØ
- FI 910 DATA &H200, &HA8A0, &H0, &HA00, &H2A80, &H0, &HA 37, &H2A80
- AA 92Ø DATA &HCØØD,&HDADD,&H3ACØ,&H7Ø77,&H7737,&H 1D4Ø,&HCØDD,&HA8ØØ
- PM 930 DATA &H3B,&H16,&H0,&H0,&H0,&H0,&H0,&H0
- ES 940 DATA &HØ, &HØ, &H1400, &HØ, &HØ, &HØ, &H55, &HØ
- B 950 DATA &HØ,&H55F1,&H4F,&HØ,&HF3Ø3,&HCF3C,&HC Ø,&HFØØ
- ₩ 960 DATA &HF30C,&HF0F0,&H0,&HF3C,&HF0AA,&H3C,& H3C00,&H820E
- HN 970 DATA &H3CB0,&H0,&H23F,&H8028,&HFC,&HF00,&H FFC0,&HF003
- BL 980 DATA &HØ, &HFBØ3, &HBFBB, &HØ, &HØ, &HEE3E, &HEC , &HØ
- N 990 DATA &H300,&H80BB,&H0,&H0,&HEE00,&H0,&H770
- PB 1000 DATA &HBB,&HDD00,&HC01D,&H5500,&H300,&H77 4,&HA240,&HBAAA
- LG 1010 DATA &HD001, &HE201, &HAAAA, &H8BAA, &H40, &HAAAA, &HAD
- B 1020 DATA &H1A00, &H820A, &HA4A0, &H0, &H4, &H0, &H1 0, &HA002
- N 1030 DATA &H14,&HD,&HA802,&H2A00,&H80BE,&HAFAA,&HAAA0,&HA0AF
- MM 1040 DATA &HAFAA,&HAAA0,&HA0AF,&HAE2A,&H2A80,&H80AA,&HAA0A,&H200
- MH 1050 DATA &HAB, &HA000, &H0, &H40, &H1, &H200, &HAB
- ሞ 1060 DATA &H26,&H12,&H2020,&H20,&H0,&H2800,&H0 ,&H0
- NP 1070 DATA &H2828,&H800,&H2028,&HA8,&H2A00,&HA0 02,&H28,&H8202
- 0K 1080 DATA &H2080, &H202, &HA80, &H2000, &H0, &H8, &H A000, &HA088
- B 1090 DATA &H8,&H8028,&H2880,&H0,&H88A2,&HAA0,& H200,&H808A
- F6 1100 DATA &HA000, &H2800, &H0, &H2000, &H0, &H200, & H8000. &H0
- NN 1110 DATA &H80A,&HA0,&H400,&H2000,&H0,&H10,&H0,&H0,&H0,&H0
- JE 1120 DATA &HE,&HA,&H1,&HC00F,&HC00E,&H3,&HB03B,&HCCCE
- BD 1130 DATA &H1, &H800A, &HA02B, &H1450, &H30CF

# **Jigsaw**

#### David Bohlke

Here's a game that demands fast thinking. "Jigsaw" generates an endless number of random puzzles for you to reassemble on your screen. But the longer you think, the lower your score. Originally written for the PCjr, the 16-color version of the program requires 128K RAM and Cartridge BASIC. We've added some simple modifications to make a 4-color version which requires only 64K RAM, and which also runs on an IBM PC with a color/graphics adapter and BASICA.

"Jigsaw" takes advantage of the IBM PCjr's 16-color, medium-resolution graphics mode to present a very pleasing color display which is also a competitive puzzle game. When you run the program, a picture puzzle of random colored blocks appears in the upper-left quarter of the screen. Just below this picture, in the lower-left part of the screen, the program draws a square piece of the picture. The remainder of the screen is dominated by a large  $8\times 8$  grid which corresponds to the puzzle. Your job is to figure out where the puzzle piece belongs in the grid. You win points for correctly placing each piece.

But there's an additional twist—time. Your score (which starts at zero) constantly decreases at a rate of one point per second. The faster you put together the puzzle, the more

points you net.

# **Taking Clues from Colors**

The timer starts ticking away as soon as the puzzle piece is fully drawn. Fitting pieces into the grid is a simple two-step process. First, move the flashing ball cursor anywhere in the grid with the cursor control keys. Position the ball on the square where you think the puzzle piece belongs, then press the Enter key. If you guessed right, the piece will be transferred to that square. If you guessed wrong, you lose ten points and another puzzle piece will appear.

When you're playing the 16-color PCjr version of Jigsaw,

you get 25 points for each puzzle piece correctly fitted into place. That means if you take longer than 25 seconds to figure out where the piece belongs, you end up losing points. There's no limit to the number of points you can lose this way. If your brain suffers a system crash and you take an hour to make up your mind, it'll cost you 3,600 points. (Of course, if you're stuck that badly, you're better off making a wild guess and

taking the 10-point penalty for guessing wrong.)

The 4-color version of Jigsaw is a lot harder. That's because the pieces of a 16-color puzzle are more likely to be different from each other than the pieces of a 4-color puzzle. Unfortunately, the current IBM PC color/graphics adapter doesn't support a 16-color mode. Even on the PCjr, the 16-color, medium-resolution mode (SCREEN 5) gobbles up so much memory that it requires a 128K machine. To compensate for the greater difficulty, the 4-color version of Jigsaw awards 50 points for each piece correctly fitted. It still subtracts one point for each second you dawdle.

Both versions of Jigsaw will test your sense of proportion. The reference picture in the upper-left part of the screen is much smaller than the actual puzzle. But puzzle pieces appear in their actual sizes. You have to scale down the piece men-

tally to figure out where it belongs in the picture.

At first it's very hard to determine where a piece fits into the grid; when the grid is blank, there are few points of reference. But like all jigsaw puzzles, it gets easier as you go along. Don't be discouraged if you have a negative score for a while. Toward the end of the game you can stack up points pretty quickly. Also, be alert for pieces which include a segment of the border—these are usually very easy to place.

If your computer is plugged into an ordinary TV set, some of the colors may bleed together and make selection more difficult. A composite color or RGB monitor shows a much sharper

display.

**Program Modifications** 

The program listing at the end of this article is the 16-color version for the 128K PCjr. To make a 4-color version for a PC with the color/graphics adapter, substitute these seven lines:

AC 10 CLEAR,,32768!:CLS
CD 20 SCREEN 1,1:COLOR 10,8:KEY OFF

CH 100 FOR I=1 TO 140: RANDOMIZE (FNTIM)

```
NC 11Ø X=RND*84+8:Y=RND*84+16:L=RND*1Ø+4:W=RND*1Ø
+4:CL=RND*3+1

MK 14Ø LINE(8,16)-(1Ø3,111),1,B:LINE(9,17)-(1Ø2,1
1Ø),1,B

DF 49Ø COLOR 1,1

LP 58Ø ZS=ZS+25:A(C,D)=1
```

### **How Jigsaw Works**

For those of you interested in BASIC programming, this list of important variables and descriptions of the major routines can be of great help. If you tire of the random rectangular puzzle pictures, you can substitute your own drawing routine.

Lines	Description						
5–32	Initialization and timer setup						
35-36	GET cursor						
50-52	Draw jigsaw grid						
100-140	Generate random puzzle						
400-490	Display single piece						
402	Gets location						
410-440	Display piece						
450-455	Prompt						
500-598	Get player's move						
500	Starts timer						
510	Displays cursor						
520-540	Read keystrokes						
545	Blanks cursor						
570	Checks if open grid						
576	Wrong grid						
580	Correct grid						
590-598	Put piece in grid						
600-622	Check for end of game						
Variables	Description						
Z(60)	Array for cursor GET,PUT						
A(8,8)	Stores grid pieces						
X,Y	Location of single piece in picture						
A,B	Location of cursor in grid						
C,D	Update of cursor location						
ZS	Score						
FNTIM	Defined timer function						
ZT	Timer						
NR	Number of pieces placed in grid						
K	Code for key pressed						

### Jigsaw

```
WH 5 DEF SEG=0:POKE 1047,64
DN 10 CLEAR,,,32768!:CLS
OC 20 SCREEN 5
              :COLOR 10,8:KEY OFF
16 30 DEFINT A-Y: DIM Z (60), A(8,8)
DL 32 DEF FNTIM=(PEEK(1132)+PEEK(1133) *256)/18.2
BF 35 CIRCLE(13,13),9,4:PAINT(12,12),4
LJ 36 GET (2,2)-(22,22), Z:CLS
CD 40 LOCATE 1,5:COLOR 2:PRINT"JIGSAW" :COLOR 1
AH 50 FOR X=120 TO 312 STEP 24:LINE(X,4)~(X,196),
     1:NEXT
CB 52 FOR Y=4 TO 196 STEP 24:LINE(120,Y)-(312,Y),
     1:NEXT
CD 100 FOR I=1 TO 70 : RANDOMIZE (FNTIM)
ND 110 X=RND*84+8:Y=RND*84+16:L=RND*20+4:W=RND*20
      +4: CL=RND*14+1
WF 120 LINE (X,Y)-(X+L,Y+W),CL,BF
ND 13Ø NEXT
KM 132 LINE(8,112)-(118,132),Ø,BF
KD 133 LINE(102,15)-(118,125),0,BF
JI 140 LINE(8,16)-(103,111),7,B:LINE(9,17)-(102,1
      10),7,B
ND 200 NR=0:A=0:B=0
NC 400 RANDOMIZE FNTIM
LA 401 LOCATE 25,1:PRINT"SCORE "; ZS\1; " ";
H 402 X=RND*7:Y=RND*7:IF A(X,Y)=1 THEN 402
#0 41Ø M=X*12+8: N=Y*12+16
EH 42Ø FOR M1=Ø TO 11:FOR N1=Ø TO 11
8N 43Ø A=POINT (M1+M, N1+N)
#C 434 LINE(16+M1#2,12Ø+N1#2)-(16+M1#2+1,12Ø+N1#2
      +1), A, BF
FJ 44Ø NEXT: NEXT
KM 450 COLOR 1:LOCATE 19,7:PRINT"USE ";
BM 451 COLOR 3:PRINT CHR$(24);CHR$(25);CHR$(26);C
      HR$(27);:COLOR 2
FA 453 LOCATE 20,1:PRINT"to move cursor";
LF 454 COLOR 2:LOCATE 21,1:PRINT"then press";
6A 455 COLOR 3:LOCATE 22,5:PRINT"ENTER ...";
ME 49Ø COLOR 10,0
HF 500 POKE 1052, PEEK (1050): ZT=FNTIM: A=RND*7: B=RN
      D*7
FC 51Ø PUT (A*24+12Ø, B*24+4), Z
AE 512 ZS=ZS-(FNTIM-ZT): ZT=FNTIM
# 515 LOCATE 25,1:PRINT"SCORE "; ZS\1; " ";
B6 520 K=0:A$=INKEY$:IF LEN(A$)>1 THEN K=ASC(MID$
      (A$,2)) ELSE IF LEN(A$)=1 THEN K=ASC(A$)
PK 53Ø IF K=72 THEN D=B-1: IF D<Ø THEN D=7
LD 532 IF K=BØ THEN D=B+1: IF D>7 THEN D=Ø
```

```
JH 534 IF K=77 THEN C=A+1:IF C>7 THEN C=Ø
MI 536 IF K=75 THEN C=A-1: IF C<Ø THEN C=7
MN 540 IF K=13 THEN PUT (A*24+120, B*24+4), Z:GOTO 5
      7Ø
CH 545 PUT (A*24+120,B*24+4),Z:A=C:B=D:GOTO 510
AA 57Ø IF A(C,D)=1 THEN 51Ø
MC 572 LOCATE 25,1:PRINT"SCORE "; ZS\1:" ";
FK 575 IF X=C AND Y=D THEN 580
MM 576 ZS=ZS-10: GOTO 400
LP 580 ZS=ZS+25:A(C,D)=1
NE 590 M=X*12+8: N=Y*12+16
6M 592 FOR M1=Ø TO 11:FOR N1=Ø TO 11
HG 594 A=POINT (M1+M, N1+N)
EL 596 LINE (120+C*24+M1*2, D*24+4+N1*2) - (120+M1*2+
      1+C*24, D*24+4+N1*2+1), A, BF
HM 598 NEXT: NEXT
HN 600 NR=NR+1: IF NR<64 THEN 400
FN 605 LOCATE 25,1:PRINT"SCORE ";ZS\1;
HD 610 LINE(1,116)-(110,188),5,BF
00 612 LOCATE 18,3:PRINT"Press ESC";
II 614 LOCATE 20,4:PRINT"for next";
II 616 LOCATE 22.6: PRINT"GAME";
F8 620 AS=INKEYS: IF AS="" THEN 620
HL 622 IF ASC(A$)=27 THEN RUN ELSE 620
```

# **Deadly Dungeon**

Jeff and John Klein

Step into the dark and the damp, your sword before you and a lantern dimly lit. Beware—this is an evil place.

"Deadly Dungeon" is an entertaining and exciting text adventure game for the IBM PC with color/graphics adapter and BASICA and for the PCir with Cartridge BASIC.

The opportunity has finally arrived. The chance to discover

riches and power beyond your imagination.

You enter the caverns of a dungeon with expectations. You bring with you a sword, 500 gold pieces, and a lantern. The lantern is only bright enough to illuminate an area eight feet around.

You left camp and found the old wooden door on the right. You also found the sounds of shuffling feet and the clanging of metal weapons. You smash open the door and fall into combat with six heavily armed goblins. Fast and furious, the battle's joined. You're hurt, but the goblins are on the floor. They're not moving.

There's a stack of 600 gold pieces in the corner of the room. You could go on, but you're weak and tired, so you head back to camp. You notice someone or something lurking in the shadows, but they don't bother you and you don't

bother them.

Back at camp, you ask for healing, but something is wrong. More than half your gold is gone.

Only Way Is Down

The object of "Deadly Dungeon" is to find the Dungeon Lord, eliminate him, and take over as Supreme Ruler of the Dungeon. Along the way, you encounter strange monsters, quick thieves, glittering treasure, and stairs to lower levels.

Use the cursor keys to move, and press the C key to go to camp. Camp is located at the entrance of each level and is primarily used for recovering from injuries you've sustained in

the dungeon.

You can buy ten healing points for every 1,000 gold pieces you have. To go to camp, move to the entrance of the level and press the C key. When you've discovered the stairs and take them down or up, you'll appear at the top of each level, where camp is located. From there, you must again find the stairs if you wish to return.

The game isn't easy. There are obstacles to overcome before you meet the Dungeon Lord. Not everything is in your favor.

Be careful when your opponent is stronger than you are. You have a much better chance of hitting your opponent while in combat when your strength is greater. On the other hand, if you're weaker, you stand a good chance of getting hurt.

The Dungeon's monsters get stronger as you delve into the lower levels. The doors may seem senseless at first, but they do warn you that an opponent may be on the other side.

Thieves become a problem if you're weak and need to return to camp.

Once the Dungeon Lord is found, you can either fight or run. If you decide to fight, the battle is to the death.

It's recommended that you get your strength up to these amounts when on these levels.

First level, maintain 50 strength. Second level, maintain 60 strength. Third level, maintain 70 strength.

# **Dungeon Mechanics**

Now that you know how to play, you might like to know how the program works.

On entering a level, the corridors, walls, doors, monsters, thieves, treasures, and so on, are stored as numbers in a three-dimensional array. This array is filled by the data on initialization. Using the supplied data, an entire adventure, taking place on three levels, can be launched.

Each number in the data corresponds to the contents of one  $8 \times 8$ -foot section of the dungeon. We used the numbers 0–8, which represent:

	-
/alue	Meaning
0	Corridor
1	Wall
2	Door (once broken, turns to a 3)
3	Monster (90 percent)

Value	Meaning
4	Treasure
5	Wandering Monster (50 percent)
6	Thief (10 percent)
7	Stairs
8	Dungeon Lord

#### **Build Your Own**

After you've thoroughly explored all three levels and have grown tired of the dungeon, you can easily create and explore your own. Using graph paper, box off three large rectangles to represent each level, each level consisting of a rectangle  $40 \times 21$  squares in size.

In the first row of 40 on each level, start creating the dungeon corridor at the sixteenth square. If you don't do this, camp can't be used. The rest of the layout is up to you. Make sure you put at least one staircase on each level and one Dungeon Lord in the third level.

There's no need to put a monster behind each door, because after a door is broken, the number in the array changes to 3, which represents a 90 percent chance that a monster exists on the other side. It's also a good idea to put at least 20 thieves in each level, since there's only a 10 percent chance of them stealing your money.

Here's an example of a room and corridor represented by numbers.

# **Dungeon Grid**

1	1	0	0	0	4	1	1
1	1	0	6	0	0	1	1
1	1	0	0	0	0	1	1
1	1	1	2	i	1	1	1
0	6	0	0	0	0	0	1
1	1	1	1	1	1	5	1

**Typing Deadly Dungeon** 

There are two checks to make sure you enter this program correctly. The first, "The Automatic Proofreader," should be used when you type in Deadly Dungeon. You'll find the Proofreader in Appendix B.

There's another check, though. Once you've entered and saved the program to disk, type GOTO 40000. This part of the program double-checks the data for the three levels. Data errors and their line numbers will be reported. Correct any mistakes, and type GOTO 40000. Repeat this procedure until the message Data Ok appears. (Note: This data check applies only for the data supplied in the listing below. Don't use it with your own data.)

### **Deadly Dungeon**

- 06 20 ' FOR THE PCjr WITH EXTENDED BASIC AND THE PC WITH A COLOR BOARD BOTH WITH AT LEAST 23K USABLE MEMORY
- JP 30 3
- JA 4Ø '
- LG 45 WIDTH 40:DEF SEG=61440!:IF PEEK(45534!)=253 THEN PCJR=1
- 6M 5Ø IF PCJR THEN COLOR ,#,Ø:FOR A=Ø TO 3:SCREEN Ø,1,A,Ø,1:CLS:KEY OFF:NEXT A:SCREEN Ø,1,1,1,Ø:SOUND ON ELSE SCREEN Ø,Ø,Ø
- 0E 4Ø LOCATE ,,Ø:KEY OFF:COLOR ,Ø,Ø:CLS:DEF FNZ=Z
   (X,Y,LEV):DEF FNZ1=Z(X+B,Y+A,LEV)
- %K 7Ø COLOR 12,4:LOCATE 5,13:PRINT "DEADLY DUNGED N":COLOR 2,0:LOCATE 18,12:PRINT "one moment please":COLOR 8,0:LOCATE 21,9:PRINT "be su re CapsLock is on"
- JB 80 DIM Z(40,21,3),S(9,3),B\$(9,3)
- NE 90 LEV=1: YS=50: TR=500
- H 100 FOR C=1 TO 3:FOR A=1 TO 21:READ A\$:FOR B=1
  TO 40:Z(B,A,C)=VAL(MID\$(A\$,B,1)):NEXT B,A
- PL 110 FOR A=1 TO 9: READ S(A,C): NEXT A: FOR A=1 TO 9: READ B\$(A,C): NEXT A: NEXT C: CLS: DEF SEG=
- IA 120 LOCATE 1,16:COLOR 14,6:PRINT CHR\$(1):LOCAT
   E 23,2:COLOR 15,0:PRINT "LEVEL ";LEV;"- "
   TR"GOLD "YS"STRENGTH"
- FK 13Ø X=16: Y=1
- HO 14Ø XO=X: YO=Y
- DB 150 X=XO: Y=YO:COLOR 14,6:POKE 1050, PEEK (1052)
- PF 160 A\$=INKEY\$: IF A\$="" THEN A=RND: GOTO 160

```
FF 170 A$=RIGHT$(A$,1):IF A$<>"H" AND A$<>"P" AND
        A$<>"K" AND A$<>"M" AND A$<>"C" THEN 160
FO 180 IF AS="C" AND Y=1 AND X=16 THEN 1300
th 190 IF A$="P" AND Y<21 THEN Y=Y+1
M 200 IF A$="H" AND Y>1 THEN Y=Y-1
0J 21Ø IF A$="M" AND X<4Ø THEN X=X+1
EC 220 IF AS="K" AND X>1 THEN X=X-1
₩ 23Ø IF FNZ=-1 THEN 36Ø
KC 240 IF FNZ=1 THEN SOUND 70,.05:60TO 150
KJ 25Ø IF FNZ>1 OR FNZ<-2 THEN GOTO 5ØØ
EH 260 FOR A=-1 TO 1:FOR B=-1 TO 1
EN 270 IF A+Y=0 OR A+Y=22 OR B+X=0 OR B+X=41 THEN
       33Ø
HB 280 IF FNZ1=1 THEN 330
© 290 IF FNZ1=2 THEN Z(X+B,Y+A,LEV)=-2.5
ID 300 IF FNZ1=0 THEN Z(X+B,Y+A,LEV)=-2
KB 310 IF FNZ1>3 THEN Z(X+B,Y+A,LEV)=-FNZ1
II 320 LOCATE Y+A, X+B:PRINT " "
IJ 33Ø NEXT B, A
₩ 340 IF FNZ=-6 THEN Z(X,Y,LEV)=-6.5
PJ 350 IF FNZ<>-6 AND FNZ<>-6.5 THEN Z(X,Y,LEV)=-
IN 360 LOCATE YO, XO:PRINT " ":LOCATE Y, X:PRINT CH
      R$(1):GOTO 140
NH 500 DEF SEG=0:POKE 1050,PEEK(1052):Z1=0: C=INT
      (ABS(FNZ))-1:LOCATE YO, XO:FRINT " ":LOCATE
       Y, X: PRINT CHR$(1)
W 510 ON C GOSUB 400,700,800,900,1000,1100,1200
M 515 IF Z1=9 THEN 120
CB 520 IF Z1=0 THEN A=X0:X0=X:X=A:A=Y0:Y0=Y:Y=A
NN 53Ø LOCATE 23,2:COLOR 15,0,0:PRINT "LEVEL ";LE
      V; "- "TR"GOLD "YS"STRENGTH"" ";:LOCATE
       24,1:PRINT SPC(39);:COLOR 14,6:LOCATE YO,
      XO:PRINT " ":LOCATE Y, X:PRINT (CHR$(1)
FI 540 IF FNZ=-6.5 THEN 350
J6 550 IF Z1 THEN 260 ELSE 140
II 600 ' DOOR
NL 610 COLOR 18,0:LOCATE 24,1:PRINT " DOOR ";:COL
      OR 2,0:PRINT "-- (B) REAK (I) GNORE
F6 62Ø A$=INKEY$:IF A$="" OR A$<>"B" AND A$<>"I"
      THEN 62Ø
H6 630 IF A$="I" THEN Z(X,Y,LEV) =-2.5 ELSE LOCATE
       24,1:PRINT " DOOR BROKEN"SPC(19);:Z(X,Y,L
      EV)=3:IF PCJR THEN FOR A=14 TO Ø STEP -2:N
      DISE 6, A, 1.5: NEXT A: FOR A=1 TO 800: NEXT A
NI 640 RETURN
KH 700 ' MONSTERS
```

NK 705 IF FNZ<-4 OR FNZ>-3 THEN IF RND>.9 THEN 2(

X.Y.LEV) = Ø: GOTO 795

- PA 710 D=INT(RND\*9)+1:IF FNZ>-4 AND FNZ<-3 THEN D = (-FNZ-3) \*10
- FO 715 C\$=B\$(D, LEV):MS=S(D, LEV)
- NJ 720 FOR A=1 TO 2:SOUND 120,4:SOUND 70,4:NEXT A :FOR A=1 TO 400:NEXT A:FOR A=14 TO 0 STEP -1:NEXT A:IF PCJR THEN SCREEN 0,1,0,0,0
- JF 725 COLOR 15,0,4:CLS:LOCATE 1,10:PRINT "YOU ME ET A MONSTER/S":LOCATE 2,10:PRINT STRING\$( 20,205):LOCATE 4,5:PRINT "MONSTER: "C\$:LOC ATE 6,5:PRINT "STRENGTH: "MS:LOCATE 9,5:PR INT "YOUR STRENGTH: "YS:LOCATE 13.8
- NK 73Ø IF FNZ=-8 THEN 122Ø
- F8 735 CT=Ø: OMS=INT (MS/4.5)+1
- OH 740 IF CT=OMS THEN 780
- MB 745 LOCATE 6,15:PRINT CINT(MS)" ":LOCATE 9,20: PRINT YS" ":IF Z1=9 THEN 765 ELSE LOCATE 1 2,12:PRINT "(F)IGHT OR (R)UN":DEF SEG=0:PO KE 1050,PEEK(1052)
- IH 75Ø A\$=INKEY\$:IF A\$="" OR A\$<>"R" AND A\$<>"F"
   THEN 75Ø ELSE LOCATE 12,12:PRINT SPACE\$(16
  ):IF A\$="R" AND FNZ=-8 THEN 795
- # 755 IF As="R" THEN Z(X,Y,LEV)=-3-(D/10):FOR A=
  1 TO 30:SOUND 70,.07:FOR B=1 TO 50:NEXT B,
  A:GOTO 795
- LA 760 IF FNZ=-8 THEN Z1=9
- BH 765 IF RND\*2Ø+1+(YS-MS)/12>1Ø THEN MS=MS-4.5:L OCATE 15,15:PRINT "YOU HIT ":CT=CT+1:FOR A=15 TO Ø STEP -1:NEXT A ELSE LOCATE 15,1 5:PRINT "YOU MISSED"
- HC 770 IF RND\*20+1+(MS-YS)/12>10 THEN YS=YS-4:LOC ATE 17,15:PRINT "HE HIT ":FOR A=15 TO 0 STEP -1:NEXT A ELSE LOCATE 17,15:PRINT "HE MISSED"
- LA 775 IF YS<1 THEN 1400 ELSE 740
- PK 78Ø IF FNZ=-8 THEN 123Ø
- MK 785 LOCATE 9,20:PRINT YS" ":Z(X,Y,LEV)=-2:YS=Y
  S+OMS:IF YS>100 THEN YS=100
- JE 790 LOCATE 6,15:PRINT "NONE":LOCATE 15,6:PRINT
  "YOU EARN"DMS\*4"EXPERIENCE POINTS":LOCATE
  17,9:PRINT "YOUR STRENGTH IS NOW"YS:FOR A
  =1 TO 2500:NEXT A
- 18 795 IF PCJR THEN SCREEN Ø,1,LEV,LEV,Ø ELSE GOS UB 1500
- PI 797 RETURN
- IL 800 ' TREASURE
- EC 810 A=INT(RND\*1000)+1:TR=TR+A:LOCATE 24,1:COLO R 30,4:PRINT "TREASURE";:COLOR 14,0:PRINT " -- WORTH"A"GOLD PIECES ";:FOR A=1 TO 20 00:NEXT A

- BH 820 IF FNZ<>-6.5 THEN Z(X,Y,LEV)=-2
- FN 83Ø Z1=1:RETURN
- NF 900 ' WANDERING MONSTERS
- ID 910 IF RND>.5 THEN 710 ELSE Z1=1: RETURN
- JA 1000 ' THEIF
- GN 1010 Z1=1:IF RND>.1 OR TR=0 THEN RETURN ELSE A =INT(RND\*TR/2+1):TR=TR-A
- AA 1020 COLOR 13,0:LOCATE 24,2:PRINT "THEIF STOLE D";:COLOR 29,0:PRINT A;:COLOR 13,0:PRINT "GOLD";:FOR A=1 TO 2000:NEXT A:RETURN
- JF 1100 ' STAIRS
- HN 1110 COLOR 28,5:LOCATE 24,1:PRINT "STAIRS";:CO LOR 13,0:PRINT " (I)GNORE ";:IF LEV<3 THE N PRINT "OR (D)OWN ":
- 00 1120 IF LEV>1 THEN PRINT "OR (U)P";
- DP 1130 A\$=INKEY\$: IF A\$="" THEN 1130
- JO 1140 IF A\$="I" THEN 1190
- HB 115Ø IF A\$="D" AND LEV<>3 THEN LEV=LEV+1:FOR A =1000 TO 100 STEP -100:SOUND A,2:NEXT A:G OTO 1180
- JC 116Ø IF A\$="U" AND LEV<>1 THEN LEV=LEV-1:FOR A =10Ø TO 10ØØ STEP 10Ø:SOUND A,2:NEXT A:GO TO 118Ø
- LA 117Ø GOTO 113Ø
- CC 1180 LOCATE 24,1:PRINT SPC(38);:COLOR 14,6:LOC ATE Y,X:PRINT " ":Z1=9:IF PCJR THEN SCREE N Ø,1,LEV,LEV,Ø ELSE COLOR ,Ø:CLS:GOSUB 1 500
- KI 119Ø RETURN
- LH 1200 ' DUNGEON LORD
- FN 1210 C\$="":GOTO 720
- #D 1220 LOCATE 1,3:PRINT "YOU HAVE DISCOVERED THE DUNGEON LORD":LOCATE 2,3:PRINT STRING\$(3 6,205):LOCATE 4,5:PRINT SPC(8):MS=YS:GOTO 735
- OL 1230 CLS:LOCATE 12,16:PRINT "YOU WIN"
- 88 1240 IF INKEY\$="" THEN 1240 ELSE RUN
- CE 1300 ' CAMP
- HC 1305 IF PCJR THEN SCREEN 0,1,0,0,0 ELSE CLS
- fk 1310 COLOR 15,0,1:CLS:LOCATE 2,18:PRINT "CAMP"
   :LOCATE 3,18:PRINT STRING\$(4,205):LOCATE
   6,10:PRINT "1 HEALING = 10 STRENGTH":LOCA
   TE 8,15:PRINT "COST = 1000 GOLD"
- 60 1320 COLOR 15,2:LOCATE 11,14:PRINT " YOUR STAT US ":COLOR 15,0:LOCATE 13,15:PRINT TR"GOL D":LOCATE 15,14:PRINT YS"STRENGTH":IF TR< 1000 THEN 1350
- JH 1330 LOCATE 21,6:LINE INPUT "ENTER AMOUNT OF H EALING WANTED ";A\$:A=VAL(A\$):IF A\*1000>TR THEN 1350 ELSE TR=TR-A\*1000:YS=YS+A\*10

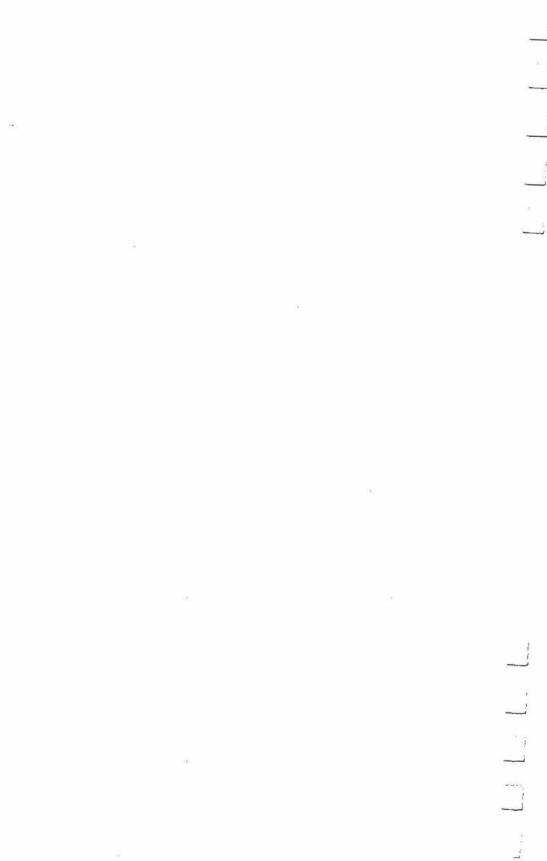
- PD 1340 LOCATE 13,15:PRINT TR"GOLD ":LOCATE 15, 14:PRINT YS"STRENGTH":GOTO 1360
- N 1350 COLOR 31,4:LOCATE 19,12:PRINT " NOT ENOUG H GOLD "
- HJ 1360 FOR A=1 TO 2500:NEXT A:IF PCJR THEN SCREE N 0,1,LEV,LEV,0 ELSE CLS:GOSUB 1500
- CN 1370 COLOR 15,0,0:GOTO 530
- MM 1400 ' YOU LOSE
- &C 1410 COLOR Ø, Ø, Ø: CLS
- #M 142Ø LOCATE 12,16:COLOR 31,Ø:PRINT "YOU LOSE":
   COL=INT(RND\*8):COLOR COL+8,COL:LOCATE RND
   \*23+1,INT(RND\*19+1)\*2:PRINT "HA";:GOTO 14
  2Ø
- MA 1500 COLOR 15,0,0:CLS:LOCATE 23,12:PRINT "one moment please":COLOR 14,6:FOR A=1 TO 21:FOR B=1 TO 40:IF Z(B,A,LEV)<0 THEN LOCATE A,B:PRINT " "
- HC 1510 NEXT B. A: RETURN
- HJ 10000 ' LEVEL 1
- BN 10010 DATA 400100001020001011000200601000020140 01000
- CM 10020 DATA 06010600161000101100011004100010100
  01000
- NF 10030 DATA 0001000020100010200000111111111110100 01400
- KL 10040 DATA 000100001010041011000100060500000111 21100
- NM 10050 DATA 00010000101111101111110121111116000 50260
- FC 10060 DATA 000140001050600000000000100010011111
- NH 10070 DATA 11211111101111121121111100010014000
- LH 10080 DATA 020006000002001000101111100010010000 10104
- FC 10090 DATA 01101111121001000100604104010010000
- EF 10100 DATA 0410140000100100010000001111111211121 10200
- FN 10110 DATA 0015100006100100010000001041500000001
- 06 10120 DATA 1110111111104100011111001001011111101 10100
- 00 10130 DATA 60101000001111100610020010616161000100 50110
- LE 10140 DATA 00201000414014000101100100101000121
- LH 10150 DATA 00101121110010012160111100202000100 01400

- CN 10160 DATA 111600000020010010110200111101000100 01000
- NK 10170 DATA 041112111111111134101111111121400100 01110
- LP 10180 DATA 000600140000107100101140000001111160 02015
- @P 10190 DATA 00000001000030230160151100000000001100 01010
- ME 10200 DATA 211111111121111100100200000060001100 01000
- 60 10210 DATA 0005006000000050200111100000000001100 41111
- L8 10220 DATA 22,10,26,13,20,21,16,6,22
- EB 10230 DATA GHOUL, PIRATE, GIANT CRAB, GIANT WORM, 3 GHOSTS, 5 GOBLINS, 2 HOBGOBLINS, 10 APES, 2 SKELETONS
- ND 20000 ' SECOND LEVEL
- CN 20010 DATA 400010414000000101111111004104140111
- EP 20020 DATA 0000100100000001000005001000100100106 00041
- KH 20030 DATA 00601001061121101121101060200100100 00001
- BG 20040 DATA 000010020010000010000101000111112112 11111
- 00 20050 DATA 11121111111012111000151000140000001# 10004
- 6K 20060 DATA 041060050060100414001020001000000010 20000
- FL 20070 DATA 0011011111011111111111111111100060010
- LM 20080 DATA 0002020001010414016000000612000000010 10060
- G! 20090 DATA 1111010041010010010111111000011111115
- LC 20100 DATA 000101111110000100101001112111060000 00001
- 0D 20110 DATA 000261004101061002020010000001011111 11161
- PK 20120 DATA 00011100010100111121001600001010002 06101
- JD 20130 DATA 00011112110100201001041001001020001 00101
- Q1 20140 DATA 06002000160121101401111001001010001 40101
- 6D 20150 DATA 00041000101101001111401121211010041 11101
- FM 20160 DATA 11111111102001401071002001006011111 50001

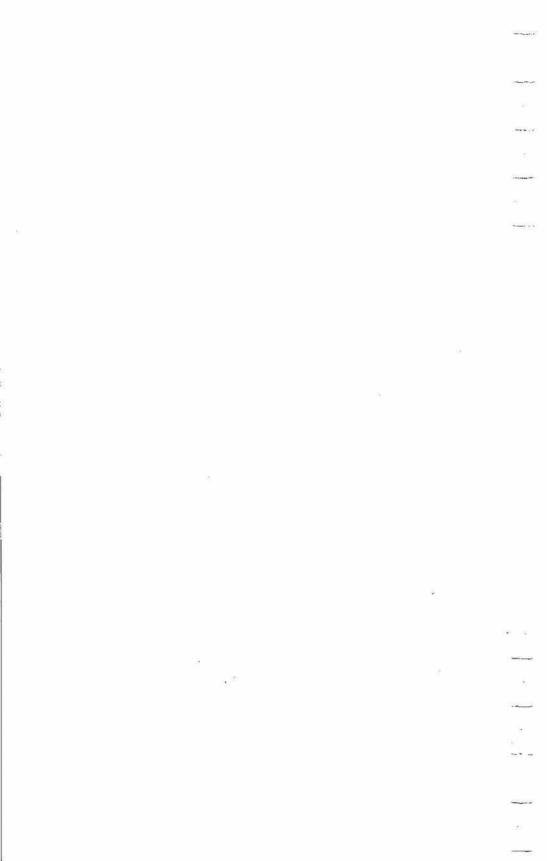
### CHAPTER TWO

- MF 20170 DATA 000000041010011110011111111111111020 01211
- IH 20180 DATA 00000600151111102041004105010060011
- JI 20190 DATA 121100001010303011110001010100000014 00000
- MO 20200 DATA 00010000020121111110020001010200000010 00600
- IJ 20210 DATA 40610000010000000600116002011100000410 00000
- MP 20220 DATA 25,30,15,17,22,28,26,35,24
- @ 20230 DATA BANSHEE, POISONOUS SPIDER, 2 GIANT RA
  T\$,3 KILLER LIZARDS, 8 GRIFFONS, 2 TROLLS,
  ZOMBIE, STEGOSAURUS, 3 CAVEMEN
- JP 30000 ' THIRD LEVEL
- 0H 30010 DATA 40000010020104101000200000160002010 20000
- P6 30020 DATA 00000010016100101000010000041000001010 10000
- QN 30030 DATA 0000000100102001016001111111100041000 10060
- JJ 30040 DATA 00060014010100101004140011121111111 12100
- PH 30050 DATA 000112111121111612111000150000006000 00104
- 53 30060 DATA 000110006000001010100000010111111112
- L 30070 DATA 0111101111111010102000060102000000410
- 00 30080 DATA 014010104104161010111111101000000010 00001
- HC 30090 DATA 610020200100101010100060001111111111
- HH 30100 DATA 01111211110010121210111111107100001
- PD 30110 DATA 0100001001001002000201000004100100000B
- 60 30120 DATA 01001110010011101011110000002001000000 01000
- MC 30130 DATA 01401140012110200020121111111112100 01600
- IB 30140 DATA 01111100020000121210602000130310100
- ## 39150 DATA 00020100611111101011111000101010100
- 11000 DATA 00016100010060001001001004161010111
- AC 30170 DATA 1111011111011111110611211111101303104 10000

- NG 30180 DATA 00020000600104102000000306031111100
- CH 30190 DATA 000111211102001011111121111111110002000 10100
- 80 30200 DATA 00041000041116010000041000000411011111 10200
- FG 30210 DATA 00001000000300010000001000000020000600
- NN 30220 DATA 38,40,44,35,33,28,39,35,42
- JO 30230 DATA BRONTOSAURUS, MEDUSA, PURPLE DRAGON, 3 POISONOUS SNAKES, 6 ZOMBIES, VAMPIRE, POLT ERGEIST, EVIL ELF, DEMON
- IN 40000 ' LEVELS DATA CHECK TYPE THIS DATA CAR EFULLY
- FH 40010 DATA 28,34,23,29,41,32,36,25,31,30,42,29,45,33,33,24,45,46,28,30,32
- FE 40020 DATA 44,24,38,23,36,39,30,39,35,24,43,31,25,39,36,45,30,47,25,24,36
- PE 40030 DATA 28,20,31,40,43,27,35,35,38,39,29,16,42,32,26,36,43,39,29,38,25
- ID 40040 DIM CHK(21,3): RESTORE 40010
- PM 40050 FOR Z=1 TO 3:FOR Y=1 TO 21:READ CHK(Y,Z)
  :NEXT Y.Z:RESTORE 10000
- CF 40060 FOR Z=1 TO 3: IF Z=2 THEN RESTORE 20000
- IM 40070 IF Z=3 THEN RESTORE 30000
- AD 40080 FOR Y=1 TO 21:READ DT\$:TOT=0:FOR X=1 TO 40:TOT=TOT+VAL(MID\$(DT\$,X,1)):NEXT X:IF CHK(Y,Z)<>TOT THEN PRINT "DATA ERROR IN" Z\*10000+Y\*10:END
- JO 40090 NEXT Y, Z:COLOR 31, Ø:PRINT "DATA OK":END



# CHAPTER THREE Education



# Type Bomb

Clark and Kathy Kidd

"Type Bomb" is a ready-to-run typing practice program for both the PC and PCjr. Multiple difficulty levels make it appropriate for a wide variety of beginning to advanced typists. The program requires BASICA and the color/graphics card on the PC or Cartridge BASIC on the PCjr.

"Type Bomb" sharpens your typing skills as you have fun. The object is to type the sentences which appear on the screen before they fall to the bottom and explode. Type Bomb will not teach touch-typing to someone unfamiliar with the keyboard, but it helps take the drudgery out of typing drills.

Each game consists of ten sentences which start at the top of the screen. The characters in each sentence must be typed *exactly*—including all spaces, numbers, and punctuation marks.

Type Bomb features eight skill levels to challenge a wide range of typists. Novices should choose skill level 1, which has simple sentences moving slowly to the bottom of the screen. As the skill levels become more advanced, the sentences begin to include numbers and other hard-to-find symbols, and the speed accelerates. Anyone who masters skill level 8 should be considered a topnotch touch-typist (say that three times real fast).

Tap Tap

After the tenth sentence has been completed, your score appears on the screen. You receive one point for each correctly typed character. Each incorrect character reduces your score by five points.

In addition, when a sentence is completely typed before it reaches the bottom of the screen and explodes, you get a ten-

point bonus.

Besides indicating your score, the screen also displays the following information at the end of each round:

- The number of correct letters typed
- The number of incorrect letters typed
- The number of sentences completed

- The number of sentences left unfinished
- The amount of time you took to complete the ten sentences

You can continue to progress within a skill level by improving your time and increasing the total points earned in a game.

#### Type Bomb

For error-free program entry, be sure to use "The Automatic Proofreader," Appendix B.

```
JB 100 REM *** TYPE BOMB
8E 11Ø NUM. COL%=4Ø
KI 120 WIDTH NUM. COL%
LA 130 KEY OFF
AF 140 CLS
HG 15Ø DEFINT A-Z
NL 155 DIM WORD. TABLE$(9), PICK. TAB(9)
BJ 160 IF NUM.COL%=80 THEN IBOTH=20 ELSE IBOTH=1
FC 170 RANDOMIZE VAL(RIGHT$(TIME$,2)) * VAL(MID$(
      TIME$, 4, 2))
MB 19Ø COLOR 10,0
GN 200 WORDS = "*THE*QUICK*BROWN*FOX*JUMPED*OYER*
      THE*LAZY*DOG*"
CF 210 I = LEN(WORD$)
EE 220 LOCATE 5, IBOTH+17: PRINT "T Y P E"
M 230 LOCATE 7, IBOTH+17: PRINT "B O M B"
BL 240 COLOR 26,0
KA 250 LOCATE 12, IBOTH+9: PRINT "PRESS ANY KEY TO
      START"
LC 255 GOSUB 10000
DL 260 COLOR 2,0
E6 \ 2700 \ J = 1
CB 280 K = 1:FOR L = 0 TO 39:GOSUB 400:NEXT
FB 300 L = 39: FOR K = 2 TO 15: GOSUB 400: NEXT
10 320 K = 15:FOR L = 0 TO 38:GOSUB 400:NEXT
IA 340 L = 0:FOR K = 2 TO 14:GOSUB 400:NEXT
HM 36Ø GOTO 28Ø
DO 400 LOCATE K, IBOTH+L
BG 41Ø PRINT MID$(WORD$, J, 1);
OF 420 J = J + 1
JH 43Ø IF J > 1 THEN J = 1
FH 440 X$ = INKEY$
CB 445 IF LEN(X$) > Ø THEN RETURN 5ØØ
NI 45Ø RETURN
AB 500 CLS
MA 510 LOCATE 2, IBOTH: PRINT "This game will help
      you practice your"
HB 520 LOCATE , IBOTH: PRINT "typing skills so you
      may become"
```

- EH 530 LOCATE , IBOTH: PRINT "a fast and accurate typist!"
- FE 540 LOCATE CSRLIN+1, IBOTH: PRINT "A phrase will appear at the top of the"
- KN 550 LOCATE , IBOTH: PRINT "screen and move towar ds the bottom."
- JB 560 LOCATE CSRLIN+1, IBOTH: PRINT "You must type the entire phrase before"
- PK 570 LOCATE ,IBOTH:PRINT "it reaches the screen bottom."
- AC 580 LOCATE CSRLIN+1, IBOTH: PRINT "The character s within the phrase must"
- EK 59Ø LOCATE ,IBOTH:PRINT "be entered in the ord er they appear,"
- LH 600 LOCATE ,IBOTH:PRINT "and must include all spaces and"
- EC 610 LOCATE , IBOTH: PRINT "punctuation marks."
- FL 612 LOCATE CSRLIN+1, IBOTH: PRINT "Each game con sists of TEN phrases."
- GA 62Ø LOCATE CSRLIN+1,IBOTH:COLOR 10,0:PRINT "SC ORING:":COLOR 2,0
- 6K 63Ø LOCATE CSRLIN+1, IBOTH: PRINT " +1 for eac h correct LETTER."
- # 64Ø LOCATE ,IBOTH:PRINT " -5 for each incorr ect LETTER."
- E 650 LOCATE , IBOTH: PRINT " +10 for each correct PHRASE."
- MA 660 COLOR 10,0
- FM 670 LOCATE CSRLIN+1, IBOTH+6: PRINT "PRESS ANY K EY TO START";
- DD 680 COLOR 2,0
- NY  $690 \times = INPUT$(1)$
- ND 700 REM \*\*\* OBTAIN SKILL LEVEL
- AF 710 CLS
- 88 720 LOCATE 3, IBOTH: COLOR 10,0: PRINT "ENTER DES IRED SKILL LEVEL:"
- M: 73Ø LOCATE CSRLIN+2, IBOTH+3: PRINT "1 ";:COLO
  R 2, Ø: PRINT "BEGINNER"
- KE 740 LOCATE CSRLIN+2, IBOTH+7:PRINT "-TD-"
- AF 750 LOCATE CSRLIN+2, IBOTH+3:COLOR 10,0:PRINT " 8 - ";:COLOR 2,0:PRINT "ADVANCED"
- M 760 X\$ = INPUT\$(1)
- CI 770 I = VAL(X\$)
- IE 780 IF I < 1 OR I > 8 THEN BEEP: GOTO 760
- KF 79Ø SKILL = 9 I
- FK 800 ON SKILL GOTO 810,820,830,840,850,860,870, 880
- HN 810 RESTORE 20100:GOTO 900
- 10 820 RESTORE 20200:GOTO 900
- JP 830 RESTORE 20300:GOTO 900

```
KA 840 RESTORE 20400: GOTO 900
LB 850 RESTORE 20500:GOTO 900
MC 860 RESTORE 20600:GOTO 900
№ 87Ø RESTORE 20700:GOTO 900
PJ 880 RESTORE 20800
OF 900 REM *** LOAD THE WORD TABLE
EP 910 FOR I = 0 TO 9
PB 920 READ WORD. TABLE$(I): NEXT
M 930 REM *** DETERMINE SELECTION ORDER
KC 94Ø FOR I=\emptyset TO 9:PICK.TAB(I) = 99:NEXT
LC 95Ø FOR I=Ø TO 9
GB 960 J = INT(RND*10)
FK 970 IF PICK. TAB(J) (> 99 THEN 960
01.980 PICK. TAB(J) = I
08 99Ø NEXT
CA 1000 REM *** CLEAR VALUES TO START THE GAME.
8A 1010 BAD. KEYS = 0
18 1020 GOOD.KEYS = 0
# 1030 GOOD.WORDS = 0
HP 1040 BAD.WORDS = 0
酬 1050 TIME$ = "00:00:00"
CH 1060 REM *** START OF MAIN GAME LOOP
88 1070 FOR L = 0 TO 9
E 1090 WORDS = WORD. TABLES (PICK. TAB(L))
IC 1100 DONES = STRINGS(LEN(WORDS), 254)
CK 1110 GOSUB 2000
PE 1120 NEXT
IG 1200 REM *** GAME OVER
MS 1210 CLS
KL 1215 X$ = INKEY$:IF LEN(X$) > Ø THEN 1215
FO 1220 LOCATE 2, IBOTH+11: COLOR 26, Ø: PRINT "G A M
             OVER"
        E
HI 1230 LOCATE 4, IBOTH: COLOR 10,0: PRINT "RESULTS:
       ":COLOR 2,Ø
@L 124Ø LOCATE CSRLIN+1, IBOTH+2: PRINT USING "Corr
       ect Letters - - - - ###":GOOD.KEYS
18 1250 LOCATE CSRLIN+0, IBOTH+2: PRINT USING "Inco
       rrect Letters - - - ###";BAD.KEYS
00 1260 LOCATE CSRLIN+0, IBOTH+2:PRINT USING "Comp
       lete Phrases - - - ###":GOOD.WORDS
01 1270 LOCATE CSRLIN+0, IBOTH+2: PRINT USING "Inco
       mplete Phrases -- ###";BAD.WORDS
PB 128Ø I = GOOD.KEYS - (BAD.KEYS * 5) + (GOOD.WO
       RDS * 10)
BK 129Ø COLOR 10.Ø
PD 1300 LOCATE CSRLIN+1, IBOTH+2: PRINT USING "FINA
       L SCORE
                           +###"; I
N 1310 LOCATE CSRLIN+3, IBOTH: COLOR 2,0:PRINT "PL
```

AY ANOTHER GAME ?"

```
FA 1320 LOCATE CSRLIN+1, IBOTH: COLOR 10,0:PRINT "
        Y = "::COLOR 2,0:PRINT "Yes"
KF 1330 LOCATE CSRLIN+1, IBOTH: COLOR 10,0: PRINT "
        N = ";:COLOR 2, Ø: PRINT "No"
N6\ 1340 \ X\$ = INPUT\$(1)
FD 1350 IF X$ = "Y" OR X$ = "y" THEN 700
LM 1360 IF X$ <> "N" AND X$ <> "n" THEN BEEP: GOTO
        1340
0K 137Ø CLS
CH 138Ø GOSUB 1ØØØØ
IC 139Ø END
IN 2000 REM *** SUBROUTINE TO MOVE SENTENCE DOWN
       THE SCREEN
N 2001 VPOS = 1
PL 2002 SPDS = 1
PL 2003 X$ = INKEY$: IF LEN(X$) > 0 THEN 2003
M 2004 CLS:LOCATE 25, IBOTH: COLOR 2,0:PRINT STRIN
       G$ (39, 24);
HK 2005 LOCATE VPOS, IBOTH
CE 2007 PRINT WORDS;
JN 2010 LOOP = 0
FJ 2020 LOOP = LOOP + 1
FH 2030 IF LOOP > SKILL * 5 GOTO 2200
N6 2040 X$ = INKEY$
00 2050 IF X$ = "" THEN 2020
PF 2060 I = ASC (X$)
KM 2070 IF I > 96 AND I < 123 THEN I = I - 32
BP 2080 X$ = CHR$(I)
IN 2100 IF X$ <> MID$(WORD$, SPOS, 1) THEN BEEP: BAD
       .KEYS = BAD.KEYS + 1:60T0 2020
## 2105 GOOD.KEYS = GOOD.KEYS + 1
M 2110 MID$(WORD$, SPOS, 1) = CHR$(254)
80 2120 LOCATE VPOS, IBOTH
80 213Ø PRINT WORD$;
PP 2140 IF WORD$ = DONE$ THEN 3000
L6 2145 LOOP = LOOP + SKILL*3
KN 2150 GOTO 2020
6F 22ØØ VPOS = VPOS + 1
PJ 221Ø IF VPOS > 24 THEN 4ØØØ
80 2220 LOCATE VPOS-1, IBOTH
HJ 2230 PRINT STRING$ (40, 32);
66 2240 LOCATE VPOS, IBOTH
CL 225Ø PRINT WORDS;
JF 226Ø GOTO 2010
KL 3000 REM *** THIS SENTENCE COMPLETE
BH 3010 FOR I = 200 TO 900 STEP 50
HS 3020 SOUND 1,2
PH 3030 NEXT
EF 3Ø35 FOR I=1 TO 35ØØ: NEXT
```

```
JK 3040 600D.WORDS = GOOD.WORDS + 1
JM 3050 RETURN
CC 4000 REM *** SENTENCE HIT BOTTOM
IL 4010 COLOR 0.7
MI 4020 CLS
PE 4022 LOCATE 24, IBOTH+4
GK 4023 COLOR 26,0
AC 4024 PRINT "
                           BOOM
PG 4026 LOCATE 23, IBOTH+4
DH 4028 PRINT "
                    * * * * * * * * * * *
00 4029 LOCATE 25, IBOTH+4
DD 4030 PRINT "
                    * * * * * * * * * * * *
       :: COLOR 2,0
0M + 4032 FOR I = 1 TO 15
6A 4Ø35 SOUND 4Ø,2
CF 4040 SOUND 200,1
₽0 4Ø5Ø NEXT
08 4060 FOR I = 1 TO 5000:NEXT
6P 411Ø BAD.WORDS = BAD.WORDS + 1
IR 4120 RETURN
CF 10000 REM *** PLAY A SONG
U 10100 PLAY "MB T250 L4 O3 A.GFGAAA P4 GGG P4 A
         O4 CC O3 P4 A.GFGAAAAGGAGF2"
HB 10200 RETURN
FE 20000 REM *** PHRASES FOR EACH SKILL LEVEL

    20100 REM ★★★ PHRASES FOR SKILL LEVEL 8

NE 20105 DATA "PLEASE REMEMBER TO CLOSE THE BACK
        DOOR. "
NI 20110 DATA "CALL WALT J. JOHNSON AT HIS OFFICE
         NOW!"
P8 20115 DATA "GET APPLES, PEARS, PEACHES AND ORA
        NGES."
J0 20120 DATA "SEND THE BOX TO 18290 SOUTH WASHIN
        GTON."
IH 20125 DATA "REMEMBER TO SET THE ALARM CLOCK FO
        R US."
LB 20130 DATA "TELL ROBERT HE MUST SEND THE CHECK
         NOW. "
LG 20135 DATA "I DON'T THINK IT'S NECESSARY RIGHT
         NOW. "
HK 20140 DATA "WE HAD 98% OF OUR STAFF AT THE MEE
        TING."
IL 20145 DATA "PLACE THE FILM IN BUS LOCKER # E34
        1256."
KN 20150 DATA "IT MAY SNOW TODAY IF IT GETS WARM
        SOON. "
NO 20200 REM *** PHRASES FOR SKILL LEVEL 7
```

```
HE 20205 DATA "ALL EMPLOYEES MUST BRING FORM 34-T
        . "
6N 20210 DATA "BE ADVISED THAT MR. JONES IS THERE
U 20215 DATA "CALL 367-6017 WHEN YOU FINISH SAME
JJ 20220 DATA "BIRDS ARE FOUND IN THE PARK MONDAY
FB 20225 DATA "NOW IT'S TOO LATE TO SEND THE FORM
BL 20230 DATA "SEVEN, THREE, ONE ARE PRIME VALUES
AC 20235 DATA "SELL ME SEVEN FOR $135.57 PLUS TAX
BN 20240 DATA "ERIC THOUGHT HE SHOULD ATTEND ALSO
AA 20245 DATA "DRAFT A LETTER TO SANTA CLAUS SOON
AF 20250 DATA "ARRIVE NOW IF YOU WANT A GOOD VIEW
        _ "
KK 20300 REM *** PHRASES FOR SKILL LEVEL 6
9N 2Ø3Ø5 DATA "KEEP YOUR ELEPHANT AT THE ZOO."
IK 20310 DATA "DO IT NOW OR LOSE YOUR CHANCE."
@ 20315 DATA "BRING A BOX OF FORMS WITH YOU."
PG 20320 DATA "HAVE THE GIFT WRAPPED BY THEN."
JF 20325 DATA "SAM THOUGHT HE SHOULD KEEP IT."
NF 20330 DATA "SUBTRACT 70 FROM 93 TO GET 23."
FC 20335 DATA "WEAR A WARM HAT IN THE WINTER."
KE 20340 DATA "ASK MRS. SMITH TO LEAVE TODAY."
FL 20345 DATA "THE COMPUTER IS PRINTING WORK."
KN 20350 DATA "ASK BOB TO BRING SEVEN DWARFS."
HG 20400 REM *** PHRASES FOR SKILL LEVEL 5
BE 20405 DATA "PLEASE DIAL 546-7892 NOW."
J0 20410 DATA "SEND SAM JONES THE FORMS."
JK 20415 DATA "BRING A SANDWICH AND JAM."
KH 20420 DATA "JOHN MUST MEET MY FRIEND."
AC 20425 DATA "YOU HAVE 97% ON THE TEST."
AA 20430 DATA "SEE YOUR DOCTOR AT 09:30."
8A 2Ø435 DATA "PLANT JUNE TOMATOES SOON."
CF 20440 DATA "WATCH THEIR MOVIE ON T.V."
16 20445 DATA "YOU MAY BUY ONE FOR JOHN."
PP 20450 DATA "TUNE THE RADIO TO FM 102."
FC 20500 REM *** PHRASES FOR SKILL LEVEL 4
AB 20505 DATA "SEND THE LETTER NOW."
66 20510 DATA "REMEMBER MY ADDRESS."
NG 20515 DATA "SALLY AND BOB SNORE."
UM 20520 DATA "A ZEBRA HAS STRIPES."
QL 20525 DATA "SHARPEN THAT PENCIL."
KF 20530 DATA "GET YOUR TICKET NOW."
```

CF 20535 DATA "MEET ME AT 128 MAIN."

```
00 20540 DATA "BUY A TAPE RECORDER."
OF 20545 DATA "PLEASE WAX MY FLOOR."
ID 20550 DATA "BAKE PEANUT COOKIES."
10 20600 REM *** PHRASES FOR SKILL LEVEL 3
BN 20605 DATA "SET YOUR TABLE."
IN 20610 DATA "I LIKE YOU TOO."
DK 20615 DATA "YOU MUST LEAVE."
IF 20620 DATA "SUE IS NOW SAD."
El 20625 DATA "MY DOG IS LAZY."
81 20630 DATA "BAKE YOUR CAKE."
SE 20635 DATA "WRITE THE POEM."
OE 20640 DATA "ASK HIM TO EAT."
II 20645 DATA "I LOVE POPCORN."
IP 20650 DATA "YOUR CAT IS OK."
BK 20700 REM *** PHRASES FOR SKILL LEVEL 2
JH 20705 DATA "RIGHT WAY."
01 20710 DATA "LEFT TURN."
U 20715 DATA "FORECASTS."
DG 20720 DATA "GET A PIE."
NJ 20730 DATA "STOP THAT!"
NE 20735 DATA "I WANT IT."
CE 20740 DATA "HE SAYS 5."
QN 20745 DATA "OLD TREES."
CC 20750 DATA "TWO & ONE."
NB 20755 DATA "GET $1.50."
06 20800 REM *** PHRASES FOR SKILL LEVEL 1
EN 20805 DATA RIGHT, SEVEN, STEVE, BLUES, BROWN, CHAIR
        , TABLE, LIGHT, MOONS, ZEBRA
```

## Bearmath

Gary West and Jim Bryan

This program tests youngsters on any of the four mathematical operations and has three levels of difficulty. After each set of ten problems, the program calculates a score and gives the option of another round. The article also explains two useful programming techniques: detecting a keypress in response to a screen prompt and page flipping on the PCjr. The program requires an IBM PC with BASICA and the color/graphics adapter or a PCjr with Cartridge BASIC.

"Bearmath" is a helpful math drill program if you have school-age youngsters, but the program also serves another purpose—it demonstrates a couple of handy programming techniques. It shows how to trap keystrokes in response to screen prompts (that is, menus) and how to instantly flip between two alternate screens on the PCjr. Before we dive into the program, though, let's see how you can play Bearmath.

If you have a PCjr, type in Program 1. Enter Program 1, then add the changes shown in Program 2 if you have an IBM PC. The modifications are required because the PC lacks the special screen-flipping commands found in PCjr Cartridge

BASIC.

When you run Bearmath, it first asks you (or the youngster) to type your name. Don't type in a long name (more than about nine characters), because later in the program the screen might scroll an extra line and mess up the screen formatting. Next, the program draws the face of a friendly bear on the screen. The picture is also copied to an alternate screen hidden in a safe place in memory. After you press the space bar to continue, a menu appears. By pressing a single key, you can select practice with addition, subtraction, multiplication, or division. Then the program offers a choice of three skill levels.

Once the drill begins, the problems are presented one by one. After each correct answer, the friendly bear appears. If the answer was wrong, you get a second chance. If the second response is also incorrect, the program gives the correct

answer.

Bearmath continues like this through a set of ten problems, maintaining the score at the top of the screen. After the tenth problem, the program presents three options: You can press the space bar for another set of ten problems, press E to end the program and return to BASIC, or press P to print a score report. Before pressing P, make sure your printer is connected, powered up, and online. Otherwise, the program ends and returns to BASIC. If you hit P by accident, you can exit the printing mode and return to the previous screen by pressing E.

If you press the space bar for another set of problems, the

program restarts from the beginning.

**Trapping Keystrokes** 

There are several parts of this program which may interest those who want to learn a few BASIC programming techniques. One of the most common techniques used by programmers is a routine which waits for the user to press a key to either continue the program or select a menu option.

The first menu in Bearmath is displayed by lines 32-48:

#### [Your name],

Press the number you want:

1 for addition

2 for subtraction

3 for multiplication

4 for division

IBM BASIC (and nearly any extensive Microsoft BASIC) offers two general ways to detect keypresses: INPUT and INKEY\$. In this case, since each menu option can be selected with one keystroke, it's easier to process the response with INKEY\$ instead of INPUT. With INKEY\$ only one keystroke is needed, while INPUT requires, at minimum, two keystrokes—the menu choice and the Enter key. INKEY\$ simply takes the next key pressed (or the next key in the keyboard buffer, if you pressed several keys in succession) and goes on.

INKEY\$ requires that you test for the *absence* of a keypress, too—because, if no key is pressed, the INKEY\$ function allows the program to continue as though no INKEY\$

were there.

Line 54 in Bearmath contains one example of trapping for the absence of a keypress so that only the keys you want will be accepted. The menu selections are numbered 1, 2, 3, and 4. If you press any other key, line 54 loops back to itself to prevent the program from continuing:

### 54 A\$=INKEY\$:IF A\$<>"1" AND A\$<>"2" AND A\$<>"3" AND A\$<>"4" THEN 54

Let's translate this line into English. This is a two-statement line, separated by a colon. The first statement, A\$=INKEY\$, tells the computer to read the keyboard and store any keystroke in the string variable A\$. (You could use any string variable, of course, as long as you modified the rest of the line to agree.)

The second statement (everything after A\$=INKEY\$) tests for which key was pressed. Remember that <> is the BASIC symbol for inequality, the opposite of =. In other words, line 54 tests for the negative—keypresses which aren't allowed. If it detects such a key, line 54 sends control back to line 54—repeating itself endlessly until one of the proper keys is pressed. If the key is acceptable (1, 2, 3, or 4), the program continues.

Another example of a menu with the appropriate traps can be seen in lines 61 (the menu) and 69 (the trap).

#### **Checking for Letters**

A special example of trapping is seen in lines 300, 920, and 9230. Here, the program waits until the space bar is pressed before going on to the next part of the program. For example, 9230 Q\$=INKEY\$:IF Q\$<>" "THEN 9230

Again, the first part of this two-statement line reads the keyboard with INKEY\$ and stores the keypress in a string variable, Q\$. The second statement in the line checks to see whether the keypress was not equal (<>) to a space. If the space bar is not pressed, the program repeats line 9230 endlessly. When the space bar *is* pressed, the program continues.

There aren't many complications when trapping for numbers or spaces. However, when trapping for letters, you must be more careful in building your traps. An example can be seen in lines 299–305. Line 299 gives you the option of pressing the E key to end the program, after line 296 gives you the option of pressing the space bar to continue. Line 300 accepts whatever key you press and compares it to the acceptable responses. If the pressed key is not a space bar or an E or a P, the program goes back to look for another key:

## 300 Z\$=INKEY\$:IF Z\$<>" " AND Z\$<>"E" AND Z\$<>"e" AND Z\$<>"e"

Notice that the trap includes tests for both uppercase and lowercase letters so that either is accepted. That way, it doesn't matter if the user's keyboard is in Caps Lock or standard mode.

The proper use of menus and traps can allow others who have no knowledge of the program to use it with little difficulty.

**Screen Flipping** 

One of Bearmath's features is that it displays the bear's friendly face as a small reward after each correct answer. But, normally, it takes the computer a few seconds to draw that face. If you had to wait for it to be drawn each time, you could easily become bored with the program. So Bearmath uses a different technique.

When you first run Bearmath, you'll see the face being drawn (by lines 9000–9140). After it's drawn, it's copied onto another video page. On the PCjr, Cartridge BASIC has a command that can flip to the alternate screen page instantly so the bear's face can be displayed without redrawing it each time.

Since the equivalent command is missing from BASICA on the PC, a short machine language routine was written to do the same thing, except it takes a fraction of a second longer. That's why PC users need to add the modifications in Program 2 to Program 1. The machine language routine is created in the subroutine starting at line 10000 in Program 2. Unfortunately, an explanation of how this routine works is beyond the scope of this article.

On the PCjr the technique is much easier and can be done entirely in Cartridge BASIC. First, to flip back and forth between two or more screens, the program must set aside enough video memory to hold the screens you want to use. In this particular graphics mode (SCREEN 1), each screen requires 16K of memory. (Some graphics modes require as much as 32K.) To use a second page, then, Bearmath must tell the computer to set aside another 16K of video memory. Look at line 2 in Program 1:

2 CLEAR,,,32768!:GOTO 8000

The CLEAR statement reserves a total of 32K of video memory so the program can use two pages for displays. The next statement branches to line 8000, where a subroutine asks for the user's name and then draws the bear.

#### **Active and Visual Screens**

The PCjr has two types of screen pages—the *active* page and the *visual* page. The active page is the one affected by BASIC commands which output to the screen—such as PRINT, LINE, CIRCLE, and so on. The visual page is the screen you're seeing at any moment—the screen actually displayed on the monitor.

Most of the time, the active page and the visual page are the same. But they don't have to be. When they're separated, the program can print messages or draw graphics on the active page without tipping off the user. The commands are taking effect, but invisibly to anyone looking at the monitor.

By adding extra parameters to the SCREEN statement, you can designate the active and visual pages. Line 8002 in Program 1 is an example:

#### 8002 SCREEN 1,0,0,0

This statement sets the screen to graphics mode 1 (as in SCREEN 1) and turns on the color (the first zero). The next two zeros set the active page equal to the visual page. Thus, you can see the bear's face as it's being drawn for the first time when the program starts.

After the face is drawn, line 9159 in Program 1 copies it from page 0 to page 1:

#### 9159 PCOPY 0,1

The rest is simple. When a math problem is answered correctly, the PCjr version of Bearmath displays the bear's face by just copying page 1 back to page 0:

#### 9300 PCOPY 1,0

The PCOPY command, by the way, is the one that's missing from Advanced BASIC (BASICA) on the PC.

#### Checking the Answer

Bearmath processes your answer to a math problem in line 170 and checks it in line 180. The correct answer was calculated and assigned to the variable Q earlier, in lines 500–820.

Your answer, assigned to the variable E, is subtracted from the correct answer. If the difference between your answer and the correct one is no more than 0.01, you're given credit for the problem. Then the program branches to the routine which copies the bear's face from the alternate screen page. You're told your answer was right and are given the option of pressing the space bar to continue.

Here's a brief outline of Program 1:

Lines	Description
8000-8050	Input user's name
9000-9240	Draw bear's face and copy it to other page
2-30	Setup
32-70	Print menus for operations and levels
100-111	Set up work screen
120-140	Make up problems
150-153	Branch to routine for right answer
500-520	Calculate addition answer
600-630	Calculate subtraction answer
700-720	Calculate multiplication answer
800-820	Calculate division answer
160-190	Display problem; accept and check answer
191-201	Branch to routine for right answer
9300-9350	Report that answer is correct
202-215	Give a second chance if first answer was wrong
241-250	Report if second answer was also wrong
280-310	Report score and option to continue or end
900–930	Print various prompts on the screen

#### Program 1. Bearmath (PCjr Version)

For error-free program entry, be sure to use "The Automatic Proofreader," Appendix B.

```
DB 47 PRINT TAB(5); "3 for multiplication"
HM 48 PRINT TAB(5): "4 for division"
LF 50 LINE (1,1)-(300,100),2,8
C! 54 A$=INKEY$: IF A$<>"1" AND A$<>"2" AND A$<>"
     3" AND A$<>"4" THEN 54
NJ 55 A=VAL (A$)
CM 60 LOCATE 16,5
CN 61 PRINT TAB(5); "Level 1, 2, or 3"
0L 64 LINE (1,105)-(300,140),1,B
IF 49 B$=INKEY$: IF B$<>"1" AND B$<>"2" AND B$<>"3
     " THEN 69
NP 7Ø B=VAL (B$)
J! 100 FOR N=1 TO 10
HH 101 CLS: LOCATE 2,10
KL 102 PRINT F;:PRINT " correct so far"
HK 103 LINE (1,1)-(300,20),2,B:PRINT
LM 104 LOCATE 5,15:PRINT "Problem ";N
W 105 LINE (1,23)-(300,43),1,B:PRINT
## 106 LOCATE 17,2:PRINT NME$; ", "
NK 107 LINE (1,115)-(300,140),1,B
01 110 LOCATE 20,2:PRINT "Type your answer and pr
      ess Enter."
LN 111 LINE (1,145)-(300,165),2,8
EJ 120 LET W=10^B
F0 13Ø LET C=INT(RND(1) *W)+1
GK 131 LET D=INT(RND(1) *W)+1
FG 135 TRY=1
DB 140 IF A>2 THEN LET D=(INT(D/10^(B-1)))+1
LO 150 IF A=1 THEN GOTO 500
NB 151 IF A=2 THEN GOTO 600
PE 152 IF A=3 THEN GOTO 700
BH 153 IF A=4 THEN GOTO 800
KQ 160 LOCATE 10,10:PRINT C;" ";5$;" ";D;" = ";
CP 17Ø INPUT " ",E
FK 180 IF ABS(Q-E)>.01 THEN GOTO 202
JC 19Ø PRINT
PC 191 GOTO 9300
60 200 LET F=F+1
FM 201 GOTO 260
U 202 PRINT: PRINT: IF TRY=2 THEN 241
ME 203 PRINT TAB(15); "Incorrect"
₩ 204 PRINT:PRINT:TRY=2
KC 205 LINE (1,90)-(300,108),1,B
CE 207 PRINT TAB(15); NME$; ", ": PRINT TAB(15); "try
      again."
PK 208 LINE (1,115)-(300,140),2,8
N 210 LOCATE 20,2:PRINT "Type your answer and pr
      ess Enter."
M 212 LOCATE 10,2:PRINT BLANK$
FA 215 GOTO 160
```

- NG 241 PRINT TAB(15); "Incorrect"
- IH 243 PRINT:PRINT:PRINT TAB(2);BLANK\$:PRINT TAB(
  2);BLANK\$
- CH 244 LOCATE 16,10:PRINT "The correct answer is" :PRINT TAB(18):Q
- LK 245 LINE (1,90)-(300,108),1,B
- PM 246 LINE (1.115)-(300.140).2.B
- JL 250 GOSUB 900
- AG 26Ø NEXT N
- BM 27Ø CLS
- LP 280 LOCATE 3,12:PRINT "Your score is"
- KF 281 LOCATE 4,12:PRINT F; " OUT OF 10"
- CK 282 LINE (1,1)-(300,40),1,B
- PC 29Ø FOR M=1 TO 1ØØ
- AF 291 NEXT M
- M 295 PRINT: PRINT: PRINT
- 8H 296 PRINT TAB(10); "Press spacebar ":PRINT TAB(
  17); "for"
- JO 297 PRINT TAB(10); "next 10 problems"
- IN 298 LINE (1,50)-(300,85),2,8
- NB 300 Z\$=INKEY\$:IF Z\$<>" "AND Z\$<>"E" AND Z\$<>" e" AND Z\$<>"P" AND Z\$<>"p" THEN 300
- EH 305 IF Z\$="e" OR Z\$="E" THEN CL5:LOCATE 12,16: PRINT "Goodbye!":LOCATE 22,20:END
- HO 310 IF Z\$=" " THEN RUN
- JK 32Ø CLS:LOCATE 2,3:PRINT "Please be sure that printer is on.":LINE (1,1)-(300,20),1,B
- FI 325 LOCATE 5,4:PRINT "Press P to continue printing.":LOCATE 7,6:PRINT "Press E to exit printing.":LINE (1,25)-(300,65),2,B
- 6K 33Ø PT\$=INKEY\$:IF PT\$<>"P" AND PT\$<>"p" AND PT
  \$<>"E" AND PT\$<>"e" THEN 33Ø
- LN 335 IF PT\$="E" OR PT\$="e" THEN 270
- CA 340 FOR X=1 TO 60:LPRINT "+";:NEXT X:LPRINT "
  ":LPRINT:LPRINT
- HO 345 LPRINT TAB(SPOT+10); NME\$:
- PA 350 LPRINT TAB(20); "worked with operation ":LP RINT TAB(30-.5\*LEN(OP\$)); OP\$:LPRINT TAB(24); " at level ":B\$
- JL 36Ø LPRINT:LPRINT TAB(13); "and worked ";F;" ou
  t of 1Ø problems.":LPRINT:LPRINT:LPRINT TA
  B(35); "The bear":LPRINT TAB(35); DATE\$:LPRI
  NT:FOR X=1 TO 6Ø:LPRINT "+";:NEXT X:LPRINT

```
50 37Ø GOTO 27Ø
KA 500 LET 5$="+"
PB 505 OF$="ADDITION"
KB 510 LET Q=C+D
E6 52Ø GOTO 16Ø
ML 600 LET 5$="-"
GH 605 OP$="SUBTRACTION"
DP 610 IF CKD THEN SWAP C.D
MM 620 LET Q=C-D
FJ 630 GOTO 160
JF 700 LET S$=" *"
BN 705 OP$="MULTIPLICATION"
JH 71Ø LET Q=C*D
FI 72Ø GOTO 16Ø
NH 800 LET S$="/"
80 805 OP$="DIVISION"
NE 810 LET Q=C/D
BD 811 IF Q<>INT(C/D) THEN C=C+1:GOTO 810
FJ 82Ø GOTO 16Ø
JI 900 PRINT
IF 901 IF NK10 THEN LOCATE 20,2:PRINT "Press spac
      ebar for next problem.
OK 910 IF N=10 THEN LOCATE 20,2:PRINT "Press spac
      ebar for your score.
DN 915 LINE (1,145)-(300,165),3,8
FB 920 D$=INKEY$:IF D$<>" " THEN 920
NJ 93Ø RETURN
MG 8ØØØ CLS
# 8001 KEY OFF:STRT=0
DN 8002 SCREEN 1,0,0,0
ON 8010 LOCATE 10,4:PRINT "Type your name and pre
       ss Enter."
IP 8020 LINE (1,50)-(300,100),1,B
N 8030 LINE (1,105)-(300,125),2,8
NM 8040 LOCATE 15,3
# 8050 INPUT " ", NME$
SM 9000 PI=3.141593
NN 9020 CLS
GK 9030 COLOR 1,3
05 9040 CIRCLE (120,50), 10, 1: PAINT (120,50), 1
PB 9045 CIRCLE (120,52),5,3:PAINT (120,52),3
UF 9050 CIRCLE (200,50),10,1:PAINT (200,50),1
MB 9055 CIRCLE (200,52),5,3:PAINT (200,52),3
NM 9060 CIRCLE (120,50),20
# 9070 CIRCLE (200,50),20
EM 9075 FOR K=148 TC 152
AA 9080 CIRCLE (160,0),K,2,1.4*PI,1.6*PI
KJ 9085 NEXT K
MN 9090 CIRCLE (160,52),50,,-1.4*PI,-1.6*PI
NE 9100 CIRCLE (160,86),100: PAINT (160,86)
```

```
M 9110 CIRCLE (160,100),50:PAINT (100,100)
PB 9120 CIRCLE (75,25),20:PAINT (75,25)
N 9125 CIRCLE (75,28),10,2:PAINT (75,28),2
6E 9130 CIRCLE (245, 25), 20: PAINT (245, 25)
9135 CIRCLE (245, 28), 10, 2: PAINT (245, 28), 2
MA 9140 CIRCLE (160,52),50,0,-1.4*PI,-1.6*FI
HH 9142 LOCATE 4,2:PRINT "B":LOCATE 6,2:PRINT "E"
       :LOCATE 8,2:PRINT "A":LOCATE 10,2:PRINT "
       R"
FB 9143 LOCATE 12,2:PRINT "M":LOCATE 14,2:PRINT "
       A":LOCATE 16,2:PRINT "T":LOCATE 18,2:PRIN
       T "H"
PB 915Ø PAINT (160,100),0
JB 9155 LINE (1,1)-(300,170),2,B
M 9156 LINE (20,1)-(20,170),2
MP 9159 PCOPY Ø. 1
JN 9160 FOR Q=1 TO 5
FI 9170 COLOR Q,Q:FOR P=1 TO 150:NEXT P
BK 918Ø BEEP
NM 919Ø NEXT Q
FF 9200 COLOR 1,3
M 9210 IF N<10 THEN LOCATE 23,7:PRINT "Press spa
       cebar to go on.
MK 9211 IF N=10 THEN LOCATE 23,7:PRINT "Press spa
       cebar for score.
EC 9220 LINE (1,173)-(300,187),2,8
PH 9230 Q$=INKEY$: IF Q$<>" " THEN 9230
CA 9235 IF STRT=Ø THEN STRT=1:GOTO 6
PJ 924Ø GOTO 2ØØ
10 9300 PCOPY 1,0
PE 9305 LOCATE 23,7:PRINT NME$;", you are right!"
F9 9306 LINE (1,173)-(300,187),2,B
JF 931Ø FOR X=1 TO 2
2M 932Ø BEEP
LM 9330 FOR Y=1 TO 150: NEXT Y
80 934Ø NEXT X
PD 9350 GOTO 9200
```

#### Program 2. Bearmath (Modifications for PC)

```
NO 2 GOSUB 10000:GOTO 8000
GB 6 CLS:COLOR 1,3
IJ 30 CLS:LET F=0
PF 9159 REM
CE 9235 IF STRT=0 THEN STRT=1:CALL Z:GOTO 6
JB 9240 CALL Z:GOTO 200
BC 9300 CALL Z
```

- HA 9305 LOCATE 23,7:PRINT NME\$;" you are right!
- NJ 10000 DEF SEG:ML\$=SPACE\$(39):V=VARPRT(ML\$):DEF FNML!(DUMMY)=PEEK(V+1)+256\*PEEK(V+2)
- FL 10010 RESTORE 10040:Z=FNML!(0):FOR I=0 TO 38:R EAD A:CKSUM=CKSUM+A:POKE Z+1,A:NEXT
- LE 10020 IF CKSUM=3842 THEN RETURN
- HB 10030 SCREEN 0,0,0:COLOR 31:PRINT"Error";:COLO R 7:PRINT" in DATA statements.":END
- N 10040 DATA 85,30,190,0,0,187,0,16,142,219,139, 4,187,0,184,142,219,135,4,187,0,16,142,2
- II 10050 DATA 137,4,70,70,129,254,0,64,114,227,31 ,93,202,0,0

# Skyscape

Robert M. Simons
IBM PC/PCjr version by Tim Victor

This unique program, written by a planetarium director, presents the sky as it can be viewed at any date and time from the year 1977 onward—including zodiac constellations and all the visible planets. It also calculates planet tables, positions of the sun, and phases of the moon for any date and time from 1977 into the future. And if you missed Halley's Comet, "Skyscape" can show you where it was as it neared Earth in late 1985 and early 1986. Skyscape is both educational and entertaining. For the IBM PC with color/graphics adapter and the PCjr with Cartridge BASIC.

For thousands of years the sun, moon, and planets in our solar system have excited human imagination. In ancient times they were regarded as gods whose distant motions influenced the course of earthly events. Though we now understand more about the true nature of celestial objects, many facts remain unknown, and a brilliant nighttime sky still presents an inspir-

ing spectacle.

Whether you're seriously interested in the sky or just casually curious, "Skyscape" is a convenient tool for extending your knowledge. It opens a movable window on the heavens, displaying the position of our sun, moon, and neighboring planets from almost any location on Earth, at any point in time from 1977 into the distant future. Since it performs all the necessary calculations, you can enjoy and learn from this program even if you're not an astronomical expert. In addition to providing data about the position of celestial objects, it draws a sky map on the screen, showing each object as it would appear to you at the chosen location and time.

To get started, type in Skyscape and save a copy before

running it.

#### Past, Present, or Future

Skyscape begins by asking you to answer several questions. Enter the year, choosing any year from 1977 forward. In some ways this is the most important input of all, since objects in our solar system move significantly from one year to the next. After you choose the year, Skyscape allows you to enter the month and day.

Next you must enter the latitude (north/south position on Earth) from which you wish to view the sky. Latitude 0 places you, the observer, at the equator. Latitudes 1–90 place you in the northern hemisphere (north of the equator). To choose a southern latitude (south of the equator), enter a negative number from -1 to -90. Skyscape generally represents southerly locations with negative values.

To get you started, here's a short list of some cities, and the latitudes values you'd enter in Skyscape (you can easily find your location's latitude by calling the closest National Weather Service office). The actual latitudes of these cities have been rounded to the closest whole number.

City	Latitude
New York	41
Los Angeles	34
Anchorage	61
London	51
Paris	49
Athens	38
Johannesburg	-26
Peking	39
Sydney	-34
Rio de Janeiro	-23

Whenever Skyscape asks for information, it checks your entry to make sure it's in the acceptable range. If you enter an illegal value, the program displays an error message and gives you another chance.

#### The Sun and Moon

Though very different in size and composition, the sun and moon are alike in being the largest celestial objects visible from Earth. After you enter the date and latitude, Skyscape displays a table of data for the sun and moon. In addition to the date, day of the year, and latitude north or south, you'll see the following information:

Sun's geocentric angle. This figure represents the sun's position as a number of degrees relative to the vernal equinox.

The vernal equinox is where the sun is located when spring begins in the northern hemisphere (the same time that autumn begins in the southern hemisphere).

- Sun's declination. The number of degrees north or south of the equator. Negative values indicate a southerly location.
- Sun's altitude at noon. The location of the sun in degrees from the northern or southern horizon at noon.
- Sun's right ascension. Just as longitude and latitude indicate locations on the Earth, *right ascension* and *declination* are used to pinpoint locations in the sky. For this purpose the sky is visualized as a gigantic sphere surrounding the Earth. Declination locates a point vertically in the celestial sphere and right ascension locates it horizontally. Right ascension values are given in *hours* and *minutes* in the range 0:00–23:59. Right ascension 0:00 is exactly at the vernal equinox. Larger right ascension values lie to the east of smaller ones.
- Right ascension at 9:00 p.m. The right ascension which would be on the meridian at 9:00 p.m. This coordinate system would be found on star charts. By comparing this number with those charts, you can tell what stars and constellations would be visible at that time.
- Moon's age. The number of days since the last new moon.
- Moon's elongation. The location of the moon in degrees east or west of the sun.
- Moon's phase. The phase of the moon on this particular day.

#### The Planet Table

After viewing the sun and moon display, press P to continue to the next display screen, which contains the planet table. (Press D if you wish to enter a new date.) The planet table shows vital information about the visible planets (through Uranus, which is at the limit of our visibility). The table shows the position of each planet in right ascension and degrees east or west of the sun. It also shows the distance of each planet from Earth in millions of miles.

If you'd rather see the distance in kilometers, modify the program to change the value of ES in line 130 from 93 to 149.6 (thus, the last statement in the line should read ES=149.6).

Some planets have an asterisk to the left of the right ascension figure. This signifies that they're visible at 9:00 p.m. on the date specified. For reference, the planet table also includes the sun's present right ascension and its right ascension

at 9:00 p.m. Press D to input a new date or S to view a graphics display of the sky at any time in the current day.

The Visible Skyscape

After selecting the sky display, you must enter the hour when you wish to view the sky. The hour value should be a whole number from 0 to 23 (enter 22 for 10:00 p.m., 13 for 1:00 p.m., and so on). You'll also need to enter the minutes (0–59). Skyscape then displays the time and offers you a chance to enter different values. Press Enter when you're satisfied with the time.

Skyscape now displays the sky as it would appear at the chosen latitude, date, and time. Since the sky looks very different from different places on Earth, the latitude affects the display considerably. If your latitude is in the range 24–90 degrees north or south, the sky shows a dashed line representing the position of the celestial equator, along with symbols representing the sun, moon, and planets visible at that time. If your latitude is in the tropical region—from 23½ degrees north to 23½ degrees south—the dashed line indicates a position directly overhead.

If you're viewing in the northern hemisphere, north is above the dashed line and south is below it. In the southern hemisphere these directions are reversed. Below the sky display is a key that interprets the symbols used to represent celestial objects. If more than one object is positioned at the same spot, the symbols are displayed above each other.

At the bottom of the sky you may see two-letter abbreviations. These represent zodiac constellations that would be visible from your chosen vantage point. Skyscape uses the abbreviations AR (Aries), PI (Pisces), AQ (Aquarius), CP (Capricorn), SA (Sagittarius), SC (Scorpio), LI (Libra), VI (Virgo), LE (Leo), CA (Cancer), GE (Gemini), and TA (Taurus). Each constellation is located above the spot where its abbreviation appears. In northern latitudes, the border of each constellation's zone begins at its abbreviation and extends left. In southern latitudes, the constellation extends right from the position of its abbreviation.

Daytime skies are shown in blue and nighttime skies in black. Skyscape does not calculate the actual rising or setting time of the sun. Average rising and setting times of 6:00 a.m. and 6:00 p.m. are used in every case. You may obtain exact

rising and setting times from local newspapers. However, keep in mind that there is usually about an hour of twilight before sunrise and after sunset.

Missed Halley's?

In addition to permanent objects, Skyscape's graphics display includes Halley's Comet, which was visible during late 1985 and early 1986. If you missed the comet when it actually appeared (not that difficult to do, really), you can see where it was *supposed* to appear in the skies.

Choose a date from November 1, 1985, to May 29, 1986, and Skyscape calculates the position of Halley's Comet and includes it in the graphics display (if it would have been visible at the place and time you select). The comet's position is based on the best predictions available when this article was first written (summer 1985).

While Skyscape is generally accurate, it bases most position calculations on circular orbits. This introduces a certain element of error, since no object in our solar system has a perfectly circular orbit. The position error is most pronounced for Mercury and Mars (whose orbits are quite elliptical), but does not significantly affect other objects. I've found Skyscape accurate enough for my own purposes, which include planning astronomy classes and planetarium displays.

#### Skyscape

For error-free program entry, be sure to use "The Automatic Proofreader," Appendix B.

```
MP 100 KEY OFF: WIDTH 80: COLOR 0,0,0:CLS
OK 110 GOSUB 2210
FL 120 D$="0000310590901201511B1212243273304334":
      K1=1440:DIM HC(22):MM$="041079040"
L! 130 M$="286317345011041072102133164194225255":
      D$(1)="S":D$(2)="N":ES=93
N 140 AS="JANFEBMARAPRMAYJUNJULAUGSEPOCTNOVDEC":
      00$="OUT OF RANGE!": DG$=CHR$ (248)
HH 150 MD$="3128313031303130313031":D9=ATN(1)/4
      5: READ EE: READ M9: DIM P(6,6)
MM 160 DEF FNR(X)=INT(X*10+.5)/10
JA 170 DEF FNS(X)=INT(X*100+,5)/100
AA 180 FOR Y=1 TO 2:FOR X=1 TO 6:READ P(X,Y):NEXT
      : NEXT: Y=Ø
HM 190 FOR X=1 TO 6: READ P$(X), P(X,3): NEXT
IP 200 FOR X=1 TO 7: READ PP(X): NEXT
IK 210 J$="SATSUNMONTUEWEDTHUFRI": FOR X=1 TO 12:R
      EAD F$
```

```
FP 220 CC$=CC$+"
                     "+F$:NEXT:CC$=CC$+CC$:F$=RIG
      HT$ (CC$, 9): CC$=F$+CC$
IL 230 FOR X=1 TO 8:READ PH$(X):NEXT
JH 240 FOR X=1 TO 22: READ HC(X): NEXT: GOTO 880
JE 250 CC=MT-720: IF CC<0 THEN CC=CC+K1
MI 260 CC=CC/120:CD=CC-INT(CC):CC=INT(CC):CD=INT(
      CD*7+.2):CC=81-(CC*7+CD)
JP 270 GOSUB 2060: IF LL<0 THEN GOSUB 2610
MM 28Ø PRINT CD$: RETURN
LC 290 LOCATE 24,20: PRINT SPC(40);
EH 300 LOCATE 4, SL:PRINT "** DAYS SKY **":LOCATE
      5, SL: PRINT "----"
LH 310 LOCATE 7, SL:PRINT "INPUT THE TIME: ": LOCATE
       8, SL: PRINT "-----"
NH 320 LOCATE 9, SL: PRINT "HOUR (0-23) ":: GOSUB 245
      Ø: IF 1$<>"" THEN T1=VAL(I$)
HM 330 IF T1<0 OR T1>23 THEN LOCATE 10, SL+3: PRINT
       00$:GOTO 32Ø
PC 340 LOCATE 11, SL:PRINT "MINUTE (0-59)";:GOSUB
      2450: IF I$<>"" THEN T2=VAL(I$)
NK 350 IF T2<0 OR T2>59 THEN LOCATE 12, SL:PRINT 0
      O$:GOTO 34Ø
IA 350 R$=RIGHT$(STR$(T1),2):T$=RIGHT$(STR$(T2),2
      ): IF T2<10 THEN T$="0"+RIGHT$(T$,1)
KP 370 LOCATE 14, SL: PRINT "TIME-- "R$": "T$
CN 380 LOCATE 24, 20: GOSUB 2230: IF I$="N" THEN 290
KN 390 COLOR 3.4:CLS:T3=T1*60+T2+AA-720:IF T3<0 T
      HEN T3=T3+K1
LK 400 IF T3>K1 THEN T3=T3-K1
JL 410 MT=T3-360: IF MT<0 THEN MT=MT+K1
00 420 PT=T3+360: IF PT>K1 THEN PT=PT-K1
HN 430 LOCATE 2,18:PRINT "DAY'S SKY-- ";:GOSUB 1
      800: PRINT " "R$": "T$
JG 440 LOCATE 3, 18: PRINT"----
IH 450 COLOR 7,1:TM=VAL(R$+"."+T$):IF TM<6 OR TM>
      18 THEN COLOR 7, Ø
DB 460 XX=7+LC:FOR X=1 TO 14:LOCATE 3+X,20:IF X=X
      X THEN 48Ø
El 470 PRINT SPC (40);: GOTO 490
NB 48Ø PRINT " - - - - -
16 49Ø NEXT:LOCATE 18,20:COLOR 0,6:GOSUB 250:LOCA
TE 19,20:COLOR 7,1:IF LL<0 THEN 520</pre>
8 500 IF LL>24 THEN PRINT "E"SPC(18) "S"SPC(19) "W
      ":GOTO 540
GF 510 PRINT "UP-NORTH
                            ----OVERHEAD
                                              DOMN-
      SOUTH": GOTO 540
8) 520 IF LL<-24 THEN PRINT "E"SPC(18)"N"SPC(19)"
```

W":GOTO 540

```
66 53Ø PRINT "UP-SOUTH
                         ----OVERHEAD
                                           DOMN-
      NORTH"
PE 540 T4=AA: GOSUB 780: Y8=888
SN 550 IF Y9=999 THEN 590
PC 560 GOSUB 2460: Y8=Y9: IF A1<0 THEN 590
EM 57Ø IF U9>17 OR U9<4 THEN 59Ø
CL 580 COLOR 7,1:LOCATE U9,59-Y9:PRINT CHR$(42)
08 590 T4=AA+M2*K1: IF T4>K1 THEN T4=T4-K1
EM 600 COLOR 7,1: IF TM(6 OR TM)18 THEN COLOR 7,0
HJ 610 GOSUB 780: IF Y9=999 THEN 650
KF 620 MM=INT(M1/9.83333)+1:GOSUB 860
MK 63Ø GOSUB 246Ø: IF U9>17 OR U9<4 THEN 65Ø
0 640 LOCATE U9,59-Y9:PRINT CHR$ (MM): IF ABS (Y8-Y
      9) <=.5 THEN COLOR 1,7:LOCATE U9,59-Y9:PRIN
      T CHR$ (79): COLOR 7,1
AS 650 FOR X=1 TO 7: IF X=7 THEN 2350
JN 660 T4=P(X,6):GOSUB 780:IF Y9=999 THEN 730
18 67Ø U9=SIN((P(X,6)/4)/(1/D9)):U9=-3*U9+.5
HD 680 GOSUB 2470
KA 69Ø IF U9<4 OR U9>17 THEN 73Ø
DE 700 7=SCREEN(U9,59-Y9)
8L 71Ø IF Z<>32 AND Z<>45 THEN U9=U9+SGN(LL)+(LL=
      Ø): GOTO 700
09 720 LOCATE U9,59-Y9:PRINT CHR$(PP(X));
W 73Ø NEXT
DE 740 LOCATE 21,14: COLOR 3,4: FOR X=1 TO 5: PRINT
      CHR$(PP(X));P$(X);" ";:NEXT
JA 750 LOCATE 22, 14: PRINT "*SUN
                                  ) O (MOON
      W MOON + SUN
                   ":B$
PC 760 LOCATE 22,33:COLOR 4,3:PRINT "O":COLOR 3,4
HM 770 LOCATE 24, 20: PRINT "T- NEW TIME, P- P. TABL
      E,D- DATE,L- LAT";:SL=62:GOTO 1980
#1 78Ø Y9=999: IF MT<PT THEN 82Ø
NH 79Ø IF T4<MT AND T4>PT THEN RETURN
FK 800 IF T4<MT OR T4>K1 THEN T4=T4+K1
HM 810 GOTO 830
LB 820 IF T4<MT OR T4>PT THEN RETURN
FA 83Ø Y9=INT((T4-MT)/18+.5): IF Y9=4Ø THEN Y9=39
NK 84Ø RETURN
MC 85Ø U9=SIN((T4/4)/(1/D9)):U9=INT(-3*U9+.5):RET
      URN
FL 860 MM=VAL(MID$(MM$,3*MM-2,3)): IF L(0 AND MM<)
      81 THEN MM=ABS(MM-81)
NA 87Ø RETURN
OM 880 COLOR 0,3:CLS:LOCATE 2,20:PRINT "*******
      :PRINT "DATE INPUT":S1=Ø
CA 890 LOCATE 5,10:PRINT "----": IF Y<>0 THE
      N LOCATE 4,40:GOSUB 1800
```

- U 900 LOCATE 7,4:PRINT "YEAR";:GOSUB 2450:IF I\$
  >"" THEN Y=VAL(I\$)
- MC 910 IF Y<1977 THEN PRINT "MUST BE AFTER 1977": GOTO 900
- MB 930 IF M<1 OR M>12 THEN FRINT 00\$:GOTO 920
- N 940 DI=VAL(MID\$(MD\$,2\*M-1,2)):DI=DI-(M=2)\*LY:D
  I\$=STR\$(DI)
- 10 950 LOCATE 11,4:PRINT"DAY (1-"DI\$")";:GOSUB 24
  50:IF I\$<>"" THEN D=VAL(I\$)
- FE 960 IF D<1 OR D>DI THEN PRINT 00\$:GOT0 950
- LB 970 H\$=MID\$(A\$, (M\*3)-2,3):LOCATE 13,4:PRINT "L ATTITUDE (0-90)";:GOSUB 2450:IF I\$<>"" THE N LL=VAL(I\$)
- CH 980 GOSUB 2500
- FB 99Ø IF ABS(LL)>9Ø THEN PRINT 00\$:GOTO 97Ø
- ₽ 1000 D1=VAL(MID\$(D\$,(M\*3)-2,3))+D:GOSUB 1920:I F M>2 THEN D1=D1+LY:Y1=Y1+LY
- NA 1010 S=0:GOSUB 1540:LOCATE 4,40:GOSUB 1800:LOC ATE 5,40:PRINT "-----"
- HN 1020 LOCATE 24,20:GOSUB 2230:IF I\$="N" THEN 88
- BB 1030 LOCATE 24,20:PRINT SPC(40);
- FN 1040 D2=VAL(MID\$(M\$, (M\*3)-2,3))+D:GOSUB 1920:: IF M>2 THEN D1=D1+LY:Y1=Y1+LY
- AD 1050 D3=D2-185:IF M=3 AND D<20 THEN D2=D2+LY:D 3=D3+LY
- M 1060 IF D3<=0 THEN A=180\*D2/185:GDTO 1080
- II 1070 A=(180\*D3/(180+ZY))+180
- LC 1080 IF A<180 THEN S=23.43333\*SIN(D9\*D2\*180/185)
- LD 1090 IF A>180 THEN S=~23.43333\*(SIN(D9\*D3))
- HD 1100 IF A>=360 THEN A=A-360
- LC 1110 A=FNR(A)
- KA 112Ø S=FNR(S):A1=(SGN(LL)-(LL=Ø))\*S+9Ø-ABS(LL) :A1=FNR(A1):GOSUB 149Ø:GOSUB 144Ø
- AF 1130 W=1-(LL<0): IF A1>90 THEN A1=190-A1: W=3-W
- PE 1140 LOCATE 7,36:PRINT "DAY OF THE YEAR------"; D1
- ## 1150 LOCATE 8,36:PRINT "SUN'S GEOCENTRIC ANGLE
  ### 150 LOCATE 8,36:PRINT "SUN'S GEOCENTRIC ANGLE
  ### 150 LOCATE 8,36:PRINT "SUN'S GEOCENTRIC ANGLE
  ### 150 LOCATE 8,36:PRINT "SUN'S GEOCENTRIC ANGLE
  #### 150 LOCATE 8,36:PRINT "SUN'S GEOCENTRIC ANGLE
  ### 150 LOCATE 8,36:PRINT "SUN'S BUT'S BUT
- JH 116Ø LOCATE 9,36:PRINT "SUN'S DECLINATION--- ";STR\$(S);DG\$
- 0A 1170 LOCATE 10,36:PRINT "SUN'S ALTITUDE AT NOO N---- ";STR\$(A1);DG\$;D\$(W)
- NM 1180 LOCATE 11,36:PRINT "SUN'S RIGHT ASCENSION ----- "; A3\$
- HF 1190 LOCATE 12,36:PRINT "R.A. AT 9:00PM------"; A5\$

80 1200 LOCATE 13,36: PRINT "MOON'S AGE-----"; STR\$ (M1); "DY" HN 1210 LOCATE 14,36:PRINT "MOON'S ELONGATION----";STR\$(M8);DG\$;L\$ GM 1220 LOCATE 15,36:PRINT "MOON'S PHASE - "PH\$(M 10 1230 LOCATE 24.20: PRINT "-P- PLANET TABLE. -D-NEW DATE":: GOTO 1980 JK 1240 COLOR 7,5:CLS:LOCATE 2,20:PRINT "SKYSCAPE ";:GOSUB 1800:S1=1 MF 1250 LOCATE 4,12:PRINT"\*\* PLANET TABLE \*\*":LOC ATE 5, 12: PRINT "-----PD 1260 LOCATE 7,4:PRINT "PLANET DIST. ANG. W/ R. A. " SUN HG 1270 LOCATE 8,4:PRINT "----PO 1280 FOR X=1 TO 6:A2=Y1/P(X,2)-INT(Y1/P(X,2)): GE 1290 A2=(A2\*360)+P(X,1):IF A2>360 THEN A2=A2-3 NK 1300 E=180+A: IF E>360 THEN E=E-360 JL 1310 E1=ABS(E-A2): IF E1>180 THEN E1=360-E1 HP 1320 GOSUB 1560:E1=E1\*D9:P5=P(X,3):IF X=3 THEN GOSUB 2040 KI 1330 P(X.4)=SQR(1+P5^2-2\*P5\*COS(E1)); XX=((P5^2  $-1-P(X_4)^2/(-2*P(X_4))$ NE 134Ø P(X,5)=-ATN(XX/SQR(-XX\*XX+1))+ATN(1)\*2:P( X, 4) = INT(P(X, 4) \*93+.5) : P(X, 5) = P(X, 5) /D9JM 1350 P(X,5)=FNS(P(X,5)):Q1\$=STR\$(P(X,4)):Q2\$=S TR\$(P(X,5)) QI 1360 Q1=LEN(Q1\$):Q2=LEN(Q2\$):GOSUB 1660 FM 1370 LOCATE X+8,4:PRINT P\$(X);TAB(18-Q1);Q1\$;T AB(28-Q2); Q2\$;: IF Q3=-1 THEN PRINT DG\$"W" BE 1380 IF Q3=1 THEN PRINT DG\$"E"; KM 139Ø GOSUB 171Ø:Q4\$=STR\$(Q4):Q5\$=STR\$(Q5):IF Q 5<10 THEN Q5\$="0"+RIGHT\$(Q5\$,1) BM 1400 Q5\$=RIGHT\$(Q5\$,2):Q4\$=Q4\$+":"+Q5\$:Z=LEN(Q 4\$) 60 1410 PRINT TAB(32); QQ\$; TAB(40-Z); Q4\$: NEXT: LOCA TE 15,4:PRINT "\* - VISIBLE AT 9 P.M." JA 1420 LOCATE 17,4:PRINT "SUN'S R.A. -----";SP C(Q8); A3\$:LOCATE 18,4:PRINT "R.A. AT 7:00 PM ---"; SPC (Q9); A5\$ F8 1430 SL=52:LOCATE 24,20:PRINT "-S- FOR DAY'S S

KY, -D- FOR NEW DATE";: GOTO 1980 FD 1440 A2=K1\*A/360: IF A2>K1 THEN A2=A2-K1

3 THEN A5=A5-24

HO 1450 A3=INT(A2/60):A4=A2-A3\*60:A5=A3+9:IF A5>2

```
PM 1460 A4=INT(A2-A3*60+.5): IF A4=60 THEN A4=0:A3
       =A3+1
JO 1470 IF A3=24 THEN A3=0
GH 148Ø AA=A3*6Ø+A4:GOTO 184Ø
ER 1490 M1=((Y1/M9)-INT(Y1/M9))*M9+10:IF M1>M9 TH
       EN M1=M1-M9
PK 1500 GOSUB 2260:M8=360*M2:IF M8>180 THEN L$="W
MB 1510 IF M8<=180 THEN L$="E"
KN 1520 IF M8>180 THEN M8=360-M8
# 1530 M1=FNR(M1): M8=FNR(M8): RETURN
FL 1540 YY=INT(7*(Y1/7-INT(Y1/7))+.2):IF YY=0 THE
       N YY=7
NH 1550 K$=MID$(J$, (YY*3)-2,3):RETURN
CN 1560 Q3=0:Q1=E+180:IF Q1>360 THEN 1600
IE 1570 IF A2>E AND A2<Q1 THEN 1590
DN 1580 Q3=1:RETURN
BA 1590 Q3=-1:RETURN
LA 1600 Q1=Q1-360: IF A2<=360 AND A2>E THEN 1590
NK 1610 IF Q3<>0 THEN RETURN
EM 1620 IF A2>0 AND A2<=Q1 THEN 1590
MA 1630 IF Q3<>0 THEN RETURN
FD 1640 IF A2>Q1 THEN 1580
J6 165Ø RETURN
NK 1660 Q5=Q3*P(X,5)*4+AA:IF Q5<0 THEN Q5=Q5+K1
NN 1670 IF Q5>K1 THEN Q5=Q5-K1
EL 1680 P(X.6)=Q5:Q4=INT(Q5/60):Q5=INT(Q5-Q4*60+.
       5): IF Q5=60 THEN Q5=0:Q4=Q4+1
IH 1690 IF Q4=24 THEN Q4=0
JJ 1700 RETURN
#M 1710 SU=A5*60+A4:PS=SU+360:MS=SU-360:IF PS>K1
       THEN PS=PS-K1
MG 1720 IF MS<0 THEN MS=MS+K1
LE 1730 IF MS>PS THEN 1760
06 1740 IF P(X,6)<PS AND P(X,6)>MS THEN 1790
EI 175Ø QQ$=" ":RETURN
8K 176Ø IF P(X,6)<K1 AND P(X,6)>MS THEN 179Ø
W 1770 IF P(X,6)<PS THEN 1790
CB 178Ø GOTO 175Ø
LC 1790 QQ$="*": RETURN
BH 1800 LL$=RIGHT$(STR$(ABS(LL)),2):IF ABS(LL)<10
        THEN LL$=" "+RIGHT$(LL$,1)
#K 181Ø PRINT K$;"-- ";H$;STR$(D);",";Y;" ";LL$;D
       G$;:PRINT MID$("SN", (LL<0)+2,1);
LN 1820 IF D<10 THEN PRINT " ";
JE 1830 RETURN
MB 1940 A4$=RIGHT$(STR$(A4),2)
NP 1850 IF A4<10 THEN A4$="0"+RIGHT$(A4$,1)
NP 1860 A3$=STR$ (A3) +":"+A4$: A5$=STR$ (A5) +":"+A4$
GJ 1870 Q8=7-LEN(A3$):Q9=7-LEN(A5$):RETURN
```

- MO 1880 LY=0: IF Y/4=INT(Y/4) THEN LY=1
- 61 1890 IF Y/100=INT(Y/100) AND Y/400<>INT(Y/400)
  THEN LY=0
- FH 1900 IF Y/1000=INT(Y/1000) AND Y/4000=INT(Y/40 00) THEN LY=0
- JA 1910 RETURN
- MB 1920 Y9=Y+1: IF Y9/4=INT(Y9/4) THEN ZY=1
- 10 1930 IF Y9/100=INT(Y9/100) AND Y9/400<>INT(Y9/400) THEN ZY=0
- Q! 1940 IF Y9/1000=INT(Y9/1000) AND Y9/4000=INT(Y 9/4000) THEN ZY=0
- BF 195Ø Y1=Y-1977:Y1=Y1\*365+INT(Y1/4)+D1:IF Y<2ØØ
  Ø THEN 197Ø</pre>
- DE 1960 Y1=Y1-INT((Y-2001)/100)+INT((Y-2001)/400) -INT((Y-1)/4000)
- KC 197Ø RETURN
- JB 1980 GOSUB 2240
- JC 1990 IF 1\$="D" THEN 880
- JH 2000 IF (I\$="S" OR I\$="T") AND 51=1 THEN 290
- NK 2010 IF I\$="P" THEN 1240
- 66 2020 IF I\$="L" AND S1=1 THEN 2540
- EE 2030 GOTO 1980
- AB 2040 P5=1.376344:K5=A2\*4
- LC 2050 K5=ABS(K5-1233.73)\*90/K1:K5=K5\*D9:K5=SIN( K5)\*.3225812:P5=P5+K5:RETURN
- GM 2060 IF CC<=1 THEN CC=CC+84
- CD 2070 CD\$=MID\$(CC\$,CC-1):IF MID\$(CD\$,2,1)<>" "
  AND MID\$(CD\$,3,1)=" " THEN CD\$=" "+CD\$
- 6C 2ØBØ IF MID\$(CD\$,4Ø,1)=" " AND MID\$(CD\$,41,1)< >"" THEN CD\$=MID\$(CD\$,2)
- JF 2090 CD\$=MID\$(CD\$,2,40):RETURN
- HO 2100 DATA 356.26,29.53059,59.818184,42.719626, 262.364394,52.9196763
- 0° 2110
   DATA 134.69697,218.79464,87.97,224.7,686.98
- PO 2120 DATA 4332.79813,10759.7195,30686.5884
- NA 2130 DATA "MERCURY",.3871,"VENUS",.7233,"MARS",1.5237,"JUPITER",5.2028
- 6K 214Ø DATA "SATURN", 9.53ØB, "URANUS", 19.182
- JL 2150 DATA 4,232,229,21,237,157,231
- 0 2160 DATA "SA", "SC", "LI", "VI", "LE", "CA", "GE", "TA", "AR", "PI", "AQ", "CP"
- 00 2170 DATA "NEW", "WAXING CRESCENT", "1ST QUARTER
  ","WAXING GIBBOUS", "FULL"
- HI 2180 DATA "WANING GIBBOUS", "3RD QUARTER", "WANING CRESCENT"
- HM 2190 DATA 1770,1719,1620,1500,1418,1365,1335,1 310,1290,1275,1260
- PO 2200 DATA 1238,1220,1200,1178,1115,915,720,660,640,625,610

```
LI 2210 CLS: LOCATE 7,12: PRINT "**** SKYSCAPE ****
16 222Ø RETURN
MH 2230 PRINT "-N- TO RE-INPUT OR RETURN TO CONTI
       NUE":
EL 2240 I$="":WHILE LEN(I$) =0: I$=INKEY$: WEND: IF I
       $>"Z" THEN I$=CHR$(ASC(I$)-32)
JP 225Ø RETURN
MM 2260 M2=M1/M9: IF M1<1 OR M1>28.5 THEN M3=1
EL 2270 IF M1>=1 AND M1<6.9 THEN M3=2
IH 2280 IF M1>=6.9 AND M1<=8 THEN M3=3
DA 229Ø IF M1>8 AND M1<14.2 THEN M3=4
IC 2300 IF M1>=14.2 AND M1<15.2 THEN M3=5
OK 2310 IF M1>=15.2 AND M1<21.6 THEN M3=6
FC 232Ø IF M1>=21.6 AND M1<=22.6 THEN M3=7
DA 2330 IF M1>22.6 AND M1<=28.5 THEN M3=8
J0 2340 RETURN
JL 2350 B$="": IF Y<>1985 AND Y<>1986 THEN 730
NO 2360 IF (Y=1985 AND D1<305) OR (Y=1986 AND D1)
       149) THEN 73Ø
M 2370 HD=D1+365: IF HD>516 THEN HD=HD-365
DB 238Ø H1=(HD-295)/10:HD=INT(H1):H1=H1-HD
MA 2390 T4=HC(HD)-HC(HD+1):T4=HC(HD)-H1*T4:IF T4>
       1440 THEN T4=T4~1440
NG 2400 GOSUB 780: IF Y9=999 THEN 730
PH 2410 GOSUB 850: IF T4>1115 AND T4>1200 THEN U9=
       U9+1
HM 2420 IF T4>1290 THEN U9=U9-1
IP 2430 IF T4>615 AND T4<1115 THEN U9=U9+2
PI 2440 U(7)=U9:B$=CHR$(PP(7))+"HALLEY'S COMET":G
       OTO 68Ø
ON 2450 INPUT "": IS: RETURN
N 2460 GOSUB 850
BM 2470 IF LL>=0 THEN U9=LC+10+U9:60T0 2490
CL 248Ø U9=LC+1Ø-U9:Y9=39-Y9
KP 249Ø RETURN
IK 2500 LL$="@N": IF LL<0 THEN LL$="@S"
10 2510 L1=ABS(LL): IF ABS(LL)<24 THEN L1=40
KK 2520 LC=INT((L1-40)/7+.5):D1=VAL(MID$(D$, (M*3)
       -2,3)+D
JP 253Ø RETURN
$\text{PRINT SPC(40);}
HE 2550 LOCATE 7, SL:PRINT "NEW LATITUDE": LOCATE 8
       , SL: PRINT "----"
M 2560 LOCATE 9, SL:PRINT "LAT (0-90)";:GOSUB 245
       Ø: IF 1$<>"" THEN LL=VAL(1$)
N 2570 IF ABS(LL)>90 THEN LOCATE 10,SL+3:PRINT O
       O$:GOTO 2560
10 2580 LOCATE 24,20:GOSUB 2230:IF 1$="N" THEN 25
       40
```

1E 259Ø LOCATE 9, SL:PRINT SPC(8Ø~SL);
8K 26ØØ GOSUB 25ØØ:I\$="S":GOTO 2ØØØ
JI 261Ø CI=1:C2\$=""
DB 262Ø C1\$=MID\$(CD\$,CI,1):IF C1\$<>" " THEN 264Ø
FA 263Ø C2\$=C1\$+C2\$:CI=CI+1:GOTO 265Ø
MN 264Ø C2\$=MID\$(CD\$,CI,2)+C2\$:CI=CI+2
IC 265Ø IF CI<41 THEN 262Ø</pre>

10 2660 CD\$=C2\$: RETURN

# Hickory, Dickory, Dock

Barbara H. Schulak
IBM PC/PCjr version by Tim Victor

This fun, educational program helps children learn the concepts of telling time by relating a digital clock display to a conventional clock face. For the IBM PC (requires BASICA) and PCjr (requires Cartridge BASIC).

"Hickory, Dickory, Dock" offers an enjoyable way for children to learn how to tell time. Type in the program, then save a

copy before running it.

When you run Hickory, Dickory, Dock, it displays a round clock face as well as a digital display. Four different activities are available. The first option lets youngsters practice telling time. As the positions of the clock hands change on the screen, the digital clock display changes as well. This shows the relationship between the spatial position of hands on a clock face and the numeric representation of time.

The other three activities test a youngster's time-telling ability for hours only, hours and half-hours, or five-minute

intervals.

To move the hands to the correct position, press the 1 key to shift the minutes hand, and the 2 key to move the hours hand. Press Enter when the hands are in the right places. After five correct answers, the program plays a brief song and displays some graphics as a reward. After three incorrect choices, the program automatically moves the clock hands to the correct position.

Hit the Esc (Escape) key to return to the main menu.

#### **Hickory Dickory Dock**

For error-free program entry, be sure to use "The Automatic Proofreader," Appendix B.

PK 10 KEY OFF: RANDOMIZE TIMER

MN 2Ø DIM DD(4),DS(4,7),SS\$(1Ø),CH(12),CV(12)

的 3Ø DIM DH(5), DV(5)

FB 4Ø PI=4\*ATN(1)

HP 50 GOSUB 900:GOSUB 940:GOSUB 840

```
GA 60 EOSUB 300
KC 7Ø SCREEN 1
CH 80 GOSUB 530:GOSUB 540:GOSUB 400
CH 90 FOR CP=0 TO 3:DD(CP)=10:FOR SN=1 TO 7:DS(CP
     ,SN) =Ø: NEXT: NEXT
JM 100 HH=84:HV=36:MH=84:MV=21
H 110 IF GM<4 THEN 160
0A 12Ø HR=1:MN=Ø:GOSUB 7ØØ:GOSUB 55Ø
IA 130 LOCATE 25,11:PRINT "Press Esc to quit";
PH 140 GOSUB 500: IF K$=CHR$(27) THEN 60
SC 150 GOSUB 700:GOSUB 550:GOTO 140
PA 160 NC=0
NE 170 NW=0: HR=INT(12*RND(1))+1
LD 180 IF GM=1 THEN MN=0
PM 190 IF GM=2 THEN MN=30*INT(2*RND(1))
00 200 IF GM=3 THEN MN=5*INT(12*RND(1))
M 210 GOSUB 700: TH=HR: TM=MN:HR=1: MN=0
16 220 LOCATE 25,3:PRINT "Press Enter to answer.
      Esc to quit":
EU 23Ø K$="":WHILE K$<>CHR$(27) AND K$<>CHR$(13):
      GOSUB 550:GOSUB 500:WEND
PL 24Ø IF K$=CHR$(27) THEN 6Ø
CA 250 LOCATE 25,3: IF HR=TH AND MN=TM THEN 280
IH 260 NW=NW+1: IF NW<3 THEN PRINT SPACE$(8) "That
      's not correct"SPACE$(8)::NC=0:FOR DLAY=1
      TO 500:NEXT:GOTO 220
LB 270 HR=TH: MN=TM: GOSUB 550: PRINT SPACE$ (4) "Thi
      s is the correct answer" SPACE$(4)::FOR I=
      1 TO 1500:NEXT:GOTO 170
ID 280 PRINT SPACE$(10) "You're right!" SPACE$(11
      )::NC=NC+1:IF NC=5 THEN NC=0:GOSUB 460
$8 29Ø FOR DLAY=1 TO 15ØØ:NEXT:GOTO 17Ø
NO 300 SCREEN 0: WIDTH 40:CLS
$ 310 LOCATE 6.8:PRINT "Press key to select game
BN 320 LOCATE 8.5:PRINT "1. Hours test"
# 330 LOCATE 9,5:PRINT "2. Hours and half hours
      test"
LH 340 LOCATE 10.5: PRINT "3. Five minute interval
      s test"
H 350 LOCATE 11,5:PRINT "4. Practice"
NC 360 LOCATE 12,5:PRINT "5. Quit"
MA 370 K$=INPUT$(1):IF K$<"1" OR K$>"5" THEN 370
N 38Ø IF K$="5" THEN END
PA 390 GM=VAL(K$): RETURN
IE 400 LOCATE 1,23:PRINT " To move
DG 410 LOCATE 2,23:PRINT "this hand this key";
06 420 LOCATE 4,37: PRINT "1":LOCATE 5,37: PRINT "2
DF 43Ø LINE (196,27)-(235,28),2,BF
```

```
PA 44Ø LINE (196,35)-(235,36),1,BF
NI 45Ø RETURN
OF 460 PLAY "mnt1800318gggaaa12g14n018g14e18e14f1
      8f12e14nØ"
LC 470 PLAY "18e14c18c14e18e14d18d14a."
KA 480 PLAY "mll8gagfed14c."
NA 49Ø RETURN
F! 500 K$=INPUT$(1):IF K$="1" THEN HR=HR+1+12*(HR
ER 510 IF K$="2" THEN MN=MN+5+60* (MN=55)
MD 52Ø RETURN
OM 53Ø FOR I=1 TO 12:LOCATE CV(I), CH(I):PRINT MID
      $(STR$(I),2):NEXT:RETURN
9A 54Ø LH=241:FOR LV=128 TO 149 STEP 21:LINE (LH,
      LV) - (LH+5, LV+6), 3, BF: NEXT: RETURN
DN 560 LINE (83,75)-(HH-1,HV-1),0
NL 57Ø LINE (84,76)-(MH,MV),Ø
H 58Ø LINE (83,75)-(MH-1,MV-1),Ø
18 590 HH=84+49*SIN(PI*(HR/6+MN/360))
88 600 HV=76-40*COS(PI*(HR/6+MN/360))
CH 610 LINE (84,76)-(HH,HV),2
60 620 LINE (83.75) - (HH-1.HV-1).2
00 63Ø MH=84+63*SIN(PI*MN/3Ø)
IJ 64Ø MV=76-55*COS(PI*MN/3Ø)
OC 650 LINE (84,76) - (MH, MV),1
CD 660 LINE (83,75)-(MH-1,MV-1),1
NO 670 RETURN
± 580 LINE (83,75)-(HH-1,HV-1),2
EK 690 MH=84+63*SIN(PI*MN/30)
ME 700 CP=0:DC=INT(HR/10):IF DC=0 THEN DC=10
账 710 GOSUB 760
PL 720 CP=1:DC=HR-10*INT(HR/10):GOSUB 760
CJ 73Ø CP=2:DC=INT(MN/1Ø):GOSUB 76Ø
MM 740 CP=3:DC=MN-10*DC:GOSUB 760
NL 75Ø RETURN
IP 760 IF DD(CP)=DC THEN RETURN ELSE DD(CP)=DC
4 770 SN=0:FOR SV=0 TO 40 STEP 20:GOSUB 810:IF D
      S(CP,SN) (>HC THEN GOSUB 820:DS(CP,SN)=HC
09 78Ø NEXT
M 790 FOR SV=0 TO 20 STEP 20:FOR SH=0 TO 24 STEP
       24:GOSUB 810: IF DS(CP,SN) <>HC THEN GOSUB
      830:DS(CP,SN)=HC
IL 800 NEXT: NEXT: RETURN
FP 81Ø SN=SN+1:HC=(MID$(SS$(DC),SN,1)="1"):RETURN
10 820 PUT (174+CP*36-12*(CP)1),120+SV),DH,XOR:RE
ID 83Ø PUT (17Ø+CP*36-12*(CP>1)+SH,123+SV),DV,XOR
      : RETURN
LM 84Ø SCREEN 1:CLS
```

```
66 850 FOR LV=0 TO 3:LINE (1+(LV=1 OR LV=2),LV)-(
18-(LV=1 OR LV=2),LV),3:NEXT
L0 860 GET (0,0)-(19,3),DH:PUT (0,0),DH,XOR
L0 870 FOR LH=0 TO 3:LINE (LH,1+(LH=1 OR LH=2))-(
LH,16-(LH=1 OR LH=2)),3:NEXT

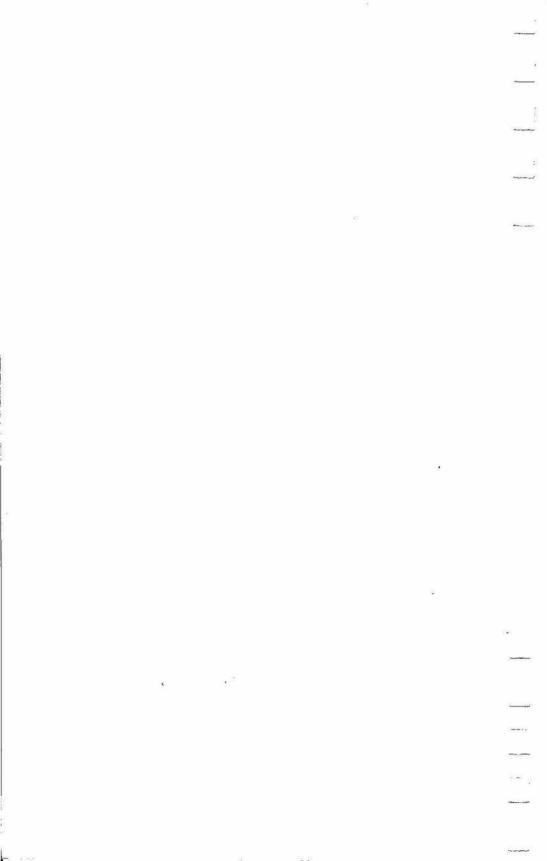
ON 880 GET (0,0)-(3,17),DV:PUT (0,0),DV,XOR
NE 890 RETURN
HB 900 FOR DC=0 TO 10:READ SS$(DC):NEXT:RETURN
BI 910 DATA "1011111","00000101","1110110","111010
1"

ON 920 DATA "0101101","1111001","1111011","100010
1"
NN 930 DATA "11111111","1111101","00000000"

940 FOR I=1 TO 12:READ CV(I),CH(I):NEXT:RETURN
NN 950 DATA 2,16,6,20,10,21
```

IG 960 DATA 14,20,18,16,19,11 FD 970 DATA 18,6,14,2,10,1 MG 980 DATA 6,2,2,6,1,11

# CHAPTER FOUR Graphics



# **Animator**

# Steve Johnson

This feature-packed utility makes it a breeze to create your own short cartoons or animation sequences on the computer screen. For the IBM PC with BASICA and color/graphics adapter, or the Enhanced Model PCjr with Cartridge BASIC.

Computer animation can be marvelous to behold but a drudge to produce. Whether you're working in BASIC or machine language, creating objects and manipulating them on the screen can mean fumbling for hours with PEEKs, POKEs, bits, bytes, and other tedious details.

"Animator" goes a long way toward automating this process. It works much like a cartoonist's sketchpad, letting you draw a series of similar images which are then displayed in rapid sequence to create the illusion of movement. Your finished cartoons can be saved on disk or tape and reloaded for viewing later.

Drawing an Image

When you run Animator, it displays an editing screen with 20 numbered frames. You can draw as many as 20 pictures, one in each frame, then flip rapidly through the frames to create animation. The frame number displayed at the upper left of the screen shows which frame you're currently working on. Normally, Animator begins the animation with frame 1 and ends with frame 20. But you can start and end the animation wherever you like. For example, a short sequence might start with frame 1 and end with frame 3. To view only part of a long sequence, you might start at frame 12 and end at frame 18, and so on. The frame number is controlled by pressing the right and left arrow keys.

The frame number also determines which frame you'll be working on when you go to the editing screen. Let's start with a simple example. Make sure the frame number is set to 1 (if it's not, change it by pressing the right or left arrow key), press the 2 key to select the editing function, and hit Enter at the next prompt. After a brief pause, Animator displays a

drawing grid with a blinking cursor. Edit mode has three main functions, selected by pressing different keys. Press D to draw with the cursor, E to erase, and M to move the cursor without disturbing anything on the screen.

Draw a simple shape on the grid to become familiar with these basic functions. As you'll see, Animator displays the shape in its actual size to the left of the drawing grid. An inverse function lets you reverse everything on the grid—every dot becomes a blank, and vice versa (be patient—it takes Animator about a minute to complete this process).

Once the picture is finished, you can press S to save it and return to the main screen. Note that you must save a picture with S to put it in the frame. If you exit the edit mode by pressing Q, the new picture is lost, and Animator uses whatever that frame previously contained. Try drawing a simple shape and saving it with S (since this is just for practice, any scribble will do). When you return to the main screen, Animator displays the picture in frame 1.

# Frame by Frame

Now you're ready to draw the next frame in the sequence. In most cases you'll want to make only slight changes from one frame to the next to simulate smooth motion. To save time, Animator lets you copy a picture from one frame to another. Let's demonstrate this by copying the picture from frame 1 to frame 2. Set the picture number to 1 with the arrow keys, then press 2 to edit. Animator displays a prompt, inviting you to enter a frame number. To edit the current picture number, you would just press Enter. However, by entering a different number you can copy the current picture into a different frame, then change that picture to make the next frame in your cartoon.

When you enter 2 at the prompt, Animator copies the picture from frame 1 into the drawing grid. When the drawing grid appears, make some change in the picture to distinguish it from frame 1. Now press S to save the picture in frame 2 and return to the main screen. Animator displays both pictures in their respective frames.

# It's Alive

After drawing a few frames, you're ready to bring them to life. The first step is to specify the starting and ending frame numbers. The starting number determines which frame begins the animation, and the ending number tells Animator where the series ends.

Set the starting number first. Use the arrow keys to set the frame number to 1, then press the 3 key. Now use the arrow keys to make the frame number match the *last* frame that contains a picture, then press the 4 key. This sets the ending number. You must always set the starting and ending numbers before selecting animation (if you don't, Animator flips through all 20 frames whether they contain pictures or not). Once these numbers are set, press the 1 key to view the sequence. Press the space bar to pause and Enter to stop it.

By selecting different speed and pause values, you can move the animated figure across the screen. The speed value can range from -15 to 15. When speed is 0, the figure is animated in place—positive values move the figure from left to right, and negative values move it from right to left. The greater the value, the faster the figure moves. Press the 5 key to decrease the animation speed and 6 to increase it.

The pause value controls the time delay between each frame of the animation. A small pause value makes the pictures change very quickly, while larger values slow down the process.

# **Macro Editing Features**

Animator provides a few macro (large-scale) editing features to help you work with longer cartoons. The insert function lets you insert a blank frame anywhere in the series. To use it, set the frame number to the number of the frame where you want to insert a blank, then press the I key. The designated picture and all those following it are bumped forward one frame. Note that the picture in frame 20 is always lost when you insert.

The delete function lets you delete any frame in the series. Change the picture number to the frame you want to eliminate, then press D. All the higher numbered pictures move down one frame, deleting the picture in the designated frame. Frame 20 is always blank after a deletion.

The inverse function (press 9) works just like inverse in editing mode, but inverts all 20 frames at once.

To clear all 20 frames, press Q to quit or C to clear. Since these last two functions can have drastic results, Animator lets you abort either one without harm.

When you finish a sequence, press S to save it on disk. The screen clears and displays three options: You can Press A to abort the save, F to list the picture files on that disk, or any other key to continue with the save. Picture filenames are limited to eight characters (the first character cannot be a number). Do not add a three-character extension—Animator automatically appends the extension .ANI when you save or load a picture file.

Finally, Animator's program option can write a separate BASIC program to display your cartoon. Press P to select this option, and sit back while Animator writes the new program to disk under the filename PRG.BAS. Afterward, Animator ends with a reminder to reload PRG.BAS and save it with a new filename. This prevents the program from being overwritten if you select this option again.

### Animator

For error-free program entry, be sure to use "The Automatic Proofreader," Appendix B.

```
# 5 DEF SEG-Ø:POKE 1047,PEEK(1047) OR 64
PK 3Ø KEY OFF: CLS: SCREEN 1: DEF SEG: POKE %H4E, 1
a 110 DEF SEG: POKE &H4E, 2
EL 150 REM ***
                MAIN PROGRAM
AF 160 CLS: SCREEN 2: KEY OFF: FOR I=1 TO 10: KEY I,"
N 170 REM *** SET UP VARIABLES
                                   ***
SD 180 DIM AX(144), BX(144), DX(144), EX(144), FX(144
      ),C%(144),G%(144),H%(144),I%(144),J%(144),
      K% (144), L% (144), M% (144), N% (144), O% (144), P%
      (144), Q% (144), R% (144), S% (144), T% (144), U% (1
      44), A(20,54): NUM=1: STA=1: EN=20: SP=0: PL=1: G
      ET(1,10)-(54,30),U%
@L 190 X=9:Y=31:LOCATE 1,39:PRINT"SPEED= ";SP::LO
      CATE 1,1:PRINT"NUMBER="; NUM;:LOCATE 1,14:P
      RINT"START=";STA;:LOCATE 1,26:PRINT"STOP="
      :EN::LOCATE 1,55:PRINT"CHANGE PAUSE=":FAST
              SET UP SCREEN
BE 200 REM ***
                               ***
PB 210 LOCATE 22,1 :PRINT"1-ANIMATE
                                        2-EDIT
         3-START PIC.
                                4-END PIC.":LOC
      ATE 23,1:PRINT"5-SLOWER
                                   6-FASTER
```

-FASTER PIC. SWITCH 8-SLOWER PIC. SWITC

- H";:LOCATE 24,1:PRINT"9-INVERSE ALL THE PI CTURES";
- 60 220 LOCATE 21,1:PRINT"(S)AVE (L)QAD (C)LEAR ALL (Q)UIT (P)R OGRAM";:LOCATE 4,78:FOR I=1 TO 10:PRINT" ";I;:NEXT:LOCATE 13,79:FOR I=11 TO 20:PR INT" ";I;:NEXT
- IH 230 LOCATE 20,1:PRINT"(I)NSERT PIC.
  (D)ELETE PIC.":
- 0K 24Ø FOR T=1 TO 2:FOR I=1 TO 5:LINE(A,X)-(A+55, Y),,B:LINE(A+6Ø,X)-(A+115,Y),,B:A=A+12Ø:NE XT:A=Ø:X=SØ:Y=1Ø2:NEXT
- 00 250 REM \*\*\* READ PICTURES \*\*\*
- DK 26Ø GET(1,1Ø)-(54,3Ø),A%:GET(61,1Ø)-(114,3Ø),B
  %:GET(121,1Ø)-(174,3Ø),C%:GET(181,1Ø)-(234
  ,3Ø),D%:GET(241,1Ø)-(294,3Ø),E%:GET(3Ø1,1Ø
  )-(354,3Ø),F%:GET(361,1Ø)-(414,3Ø),G%:GET(421,1Ø)-(474,3Ø),H%:GET(481,1Ø)-(534,3Ø),I
  %:GET(541,1Ø)-(594,3Ø),J%
- B 270 GET(1,81)-(54,101), K%: GET(61,81)-(114,101), L%: GET(121,81)-(174,101), M%: GET(181,81)-(234,101), N%: GET(241,81)-(294,101), 0%: GET(301,81)-(354,101), P%: GET(361,81)-(414,101), Q%: GET(421,81)-(474,101), R%: GET(481,81)-(534,101), S%: GET(541,81)-(594,101), T%
- LK 28Ø REM \*\*\* WAIT FOR KEY \*\*\*
- CI 290 A\$=INKEY\$:IF A\$="" THEN 290 ELSE IF LEN(A\$
  )=2 THEN 830 ELSE IF VAL(A\$)>0 AND VAL(A\$)
  <10 THEN 430
- OL 300 IF AS="P" THEN 3080
- 30 31Ø IF A\$<>"Q" THEN 35Ø
- SH 320 LOCATE 18,1:PRINT"ARE YOU SURE YOU WANT TO QUIT?"
- JB 33Ø A\$=INKEY\$:IF A\$="" THEN 33Ø ELSE IF A\$="Y"
   THEN SCREEN Ø,Ø,Ø:CLS:END ELSE LOCATE 18,
   1:PRINT STRING\$(5Ø,32):GOTO 29Ø
- ED 340 REM \*\*\* CLEAR SCREEN \*\*\*
- 88 35Ø IF A\$<>"C" THEN 39Ø
- PL 360 LOCATE 17,1:PRINT"ARE YOU SURE? <Y/N>":DEF SEG: POKE 106,0
- M 37Ø A\$=INKEY\$:IF A\$="" THEN 37Ø ELSE IF A\$="Y"
  THEN CLS:GOTO 19Ø ELSE LOCATE 17,1:PRINT
  STRING\$(66,32):GOTO 29Ø
- M 380 IF A\$="S" THEN 870 ELSE IF A\$="L" THEN 990
- 00 390 IF A\$="D" THEN 2800 ELSE IF A\$="I" THEN 23 60
- HM 400 GOTO 290
- F 410 IF FAST=0 THEN BEEP:GOTO 290 ELSE FAST=FAS T-1:LOCATE 1,58:PRINT FAST;:GOTO 290

- # 420 IF FAST=150 THEN BEEP: GOTO 290 ELSE FAST= FAST+1:LOCATE 1,68:PRINT FAST;:GOTO 290
- KP 43Ø ON VAL(A\$) GOTO 52Ø,1Ø8Ø,47Ø,5ØØ,48Ø,45Ø,4 1Ø,42Ø,3Ø4Ø
- FH 44Ø REM \*\*\* SET SPEED \*\*\*
- 01 450 IF SP=15 THEN BEEP ELSE SP=SP+1:LOCATE 1,4
- H 46Ø GOTO 29Ø
- N 470 STA=NUM:LOCATE 1,20:PRINT STA:GOTO 290
- PP 480 IF SP=-15 THEN BEEP ELSE SP=SP-1:LOCATE 1, 46:PRINT SP
- 10 49Ø GOTO 29Ø
- HP 500 EN=NUM:LOCATE 1,31:PRINT EN:GOTO 290
- LJ 510 REM \*\*\* ANIMATE PICTURES \*\*\*
- 68 530 FOR I=STA TO EN STEP DO
- CI 540 Pt1=PL:IF PL+SP<1 THEN PL=570 ELSE IF PL+S P>580 THEN PL=1
- DN 55Ø PL=PL+SP
- 0K 56Ø LOCATE 1,8:PRINT I:ON I GOTO 53Ø,64Ø,65Ø,6
  6Ø,67Ø,68Ø,69Ø,70Ø,71Ø,72Ø,73Ø,74Ø,75Ø,76Ø
  ,77Ø,78Ø,79Ø,8ØØ,81Ø,82Ø
- CN 570 FOR N=0 TO FAST\*4:NEXT:A\$=INKEY\$:IF A\$=" "
  THEN 2340 ELSE IF A\$<>"" AND LEN(A\$)<>2 T
  HEN LOCATE 1,8:PRINT NUM:LOCATE 17,1:PRINT
  STRING\$(150,32):GOTO 290
- 86 58Ø IF LEN(A\$)<>2 THEN 62Ø
- FA 590 C=ASC(RIGHT\$(A\$,1)):IF C=77 THEN SP=SP+1 E LSE IF C=75 THEN SP=SP-1
- EN 600 IF SP=-16 THEN SP=-15 ELSE IF SP=16 THEN S
- EL 610 LOCATE 1,46:PRINT SP
- €3 62Ø NEXT:GOTO 53Ø
- ₩ 630 PUT(PL1,52),U%,PSET:PUT(PL,52),A%:GOT0 570
- 0E 64Ø PUT(PL1,52),U%,PSET:PUT(PL,52),B%:GOTO 57Ø
- № 650 PUT(PL1,52),U%,PSET:PUT(PL,52),C%:GOTO 570
- M 660 PUT(PL1,52), U%, PSET: PUT(PL,52), D%: GOTO 570
- FG 670 PUT(PL1,52),U%,PSET:PUT(PL,52),E%:GOTO 570
- IM 680 PUT(PL1,52),U%,PSET:PUT(PL,52),F%:GOTO 570
- KG 690 PUT(PL1,52), U%, PSET: PUT(PL,52), G%: GOTO 570
- UF 700 PUT (PL1,52), UX, PSET: PUT (PL,52), H%: GOTO 570
- @L 710 PUT(PL1,52),U%,PSET:PUT(PL,52),I%:GOTO 570
- AN 720 PUT (PL1,52), U%, PSET: PUT (PL,52), J%: GOTO 570
- CH 73Ø PUT(PL1,52),U%,PSET:PUT(PL,52),K%:GOTO 57Ø FN 74Ø PUT(PL1,52),U%,PSET:PUT(PL,52),L%:GOTO 57Ø
- # 750 PUT (PL1,52), U%, PSET; PUT (PL,52), M%: 60T0 579

- KJ 76Ø PUT(PL1,52),U%,PSET:PUT(PL,52),N%:GOTC 57Ø
- ₩ 77Ø PUT(PL1,52),U%,PSET:PUT(PL,52),0%:GOTO 57Ø
- OF 780 PUT(PL1,52), U%, PSET: PUT(PL,52), P%: GOTO 570
- BL 790 PUT(PL1,52),U%, PSET: PUT(PL,52),0%: GOTO 570
- 00 800 PUT(PL1,52),U%,PSET:PUT(PL,52),R%:GOTO 570
- EE 810 PUT (PL1,52), U%, PSET: PUT (PL,52), S%: GOTO 570
- HK 820 PUT(PL1,52), U%, PSET: PUT(PL,52), T%: 60T0 570
- # 830 C=ASC(RIGHT\$(A\$,1)):IF C=77 THEN NUM=NUM+1 ELSE IF C=75 THEN NUM=NUM-1
- FO 840 IF NUM=0 THEN NUM=20 ELSE IF NUM=21 THEN N UM=1
- IN 850 LOCATE 1.8:PRINT NUM: GOTO 290
- PC 860 REM \*\*\* SAVE PICTURE \*\*\*
- BC 87Ø CLS
- # 880 LOCATE 10,10:PRINT"F-FILES A-ABORT SAV
  E ANY OTHER KEY TO CONTINUE"
- 8H 89Ø A\$=INKEY\$:IF A\$="" THEN 89Ø ELSE IF A\$="F"
   THEN FILES"\*.ANI" ELSE IF A\$="A" THEN GOS
   UB 2310:GOTO 19Ø
- O 900 PRINT:PRINT:PRINT:INPUT"NAME OF FILE TO SA VE";A\$:IF A\$="" THEN GOSUB 2310:GOTO 190
- JM 910 IF INSTR(A\$,".")<>0 THEN CLS:LOCATE 9,10:P RINT"NO EXTENSION PLEASE..":GOTO 890
- II 920 IF LEN(A\$)>8 THEN CLS:LOCATE 9,10:PRINT"NO
   MORE THAN 8 CHARACTERS PLEASE":GOTO 880
- © 93Ø IF VAL(RIGHT\$(A\$,1))>Ø OR RIGHT\$(A\$,1)="Ø" THEN CLS:LOCATE 9,1Ø:PRINT"THE FIRST CHAR ACTER CAN'T BE A NUMBER..":GOTO 88Ø
- BE 94Ø GOSUB 231Ø
- BL 950 A\$=A\$+".ANI"
- ## 960 DEF SEG=&HB900:BSAVE A\$,0,&H4000:PRINT"IT
  HAS BEEN SAVED. PRESS ANY KEY TO CONTI
  NUE":PRINT:PRINT:PRINT
- ## 970 A\$=INKEY\$:IF A\$="" THEN 970 ELSE CLS:GOSUB 2310:GOTO 190
- AG 980 REM \*\*\* LOAD PICTURE \*\*\*
- BH 990 CLS
- ID 1000 LOCATE 10,10:PRINT"F-FILES A-ABORT LO
  AD ANY OTHER KEY TO CONTINUE"
- CD 1010 A\$=INKEY\$:IF A\$="" THEN 1010 ELSE IF A\$="
  F" THEN FILES"\*.ANI" ELSE IF A\$="A" THEN
  GOSUB 2310:GOTO 190
- MI 1020 PRINT:PRINT:PRINT:INPUT"NAME OF FILE TO L OAD";A\$:IF A\$="" THEN GOSUB 2310:GOTO 190
- MI 1030 IF INSTR(A\$,".")<>0 THEN CLS:LOCATE 9,10: PRINT"NO EXTENSION PLEASE..":GOTO 1000
- SD 1040 IF LEN(A\$)>8 THEN CLS:LOCATE 9,10:PRINT"N D MORE THAN 8 CHARACTERS PLEASE":GOTO 100

```
PK 1050 IF VAL(RIGHT$(A$,1)) 0 OR RIGHT$(A$,1)="0
       " THEN CLS:LOCATE 9,10:PRINT"THE FIRST CH
       ARACTER CAN'T BE A NUMBER .. ": GOTO 1000
04 1060 A$=A$+".ANI":DEF SEG=&HB800:BLOAD A$,0:GO
       TO 19Ø
16 1070 REM ***
                 EDIT A PICTURE
                                  ***
JO 1080 LOCATE 16.1:PRINT"TYPE IN 21 TO ABORT":PR
       INT"NUMBER SET AT THE TOP OF THE SCREEN I
       S PIC. TO READ FROM RETURN FOR SAME":LOCA
       TE 18.1: INPUT"EDIT PICTURE NUMBER": B: IF B
       <Ø OR 8>21 THEN BEEP:GOTO 1080
JC 1090 IF B=21 THEN CLS:GOSUB 2310:GOTO 190
IA 1100 IF B=0 THEN B=NUM
ED 1110 REM *** PUT PICTURE TO EDIT ON SCREEN *
       **
PR 1120 CLS:LOCATE 1,24:PRINT"WAIT...": ON NUM GOT
       0 1130, 1140, 1150, 1160, 1170, 1180, 1190, 1200
       ,1210,1220,1230,1240,1250,1250,1270,1280,
       1290, 1300, 1310, 1320
NG 1130 PUT(1,50),A%:GOTO 1340
N 1140 PUT (1,50), B%: GOTO 1340
@M 115@ PUT(1,5@),C%:GOTO 134@
# 1160 PUT(1,50),D%:GOTO 1340
CC 1170 PUT(1,50), E%: GOTO 1340
DF 1180 PUT(1,50), F%: GOTO 1340
FI 1190 PUT(1,50), G%: GOTO
                          1740
P 1200 PUT(1,50), H%: GOTO 1340
FC 1210 PUT(1,50), IX: GOTO 1340
SF 1220 PUT(1,50), J%: GOTO 1340
H 1230 PUT(1,50), K%: GOTO 1340
JL 124Ø PUT(1,50), L%: GOTO 1340
KO 125Ø PUT(1,5Ø),M%:GOTO 134Ø
LB 126Ø PUT(1,5Ø),N%:GOTO 134Ø
ME 1270 PUT(1,50),0%:GOTO 1340
NH 128Ø PUT(1,5Ø),P%:GOTO 134Ø
PK 1290 PUT(1,50),0%:GOTO 1340
98 1300 PUT(1,50), R%: GOTO 1340
PE 1310 PUT(1,50),5%:GOTO 1340
AK 1320 PUT(1,50), TX: GOTO 1340
PO 1330 REM *** GET ON-OFF POINTS ***
J0 1340 FOR I=1 TO 20:FOR X=1 TO 54:A(I,X)=POINT(
       X, I+49)
PL 1350 NEXT: NEXT
KB 1360 REM *** DRAW EDITING SCREEN ***
IR 1370 FOR I=1 TO 20:LOCATE 3+1,14:PRINT ".....
       . . . . . . . . . . . . . . . . . .
        ......";:FOR J=1 TO 54:IF A(I,J)=1 THEN
       LOCATE 3+1,13+J:PRINT"#"
```

PE 1380 NEXT: NEXT

```
88 1390 LOCATE 1.30:PRINT"(0)UIT (D)RAW
       E
           (E) RASE (C) LEAR (S) AVE (I) VERSE
FL 1400 GOTO 1550
OF 1410 REM *** PLACE CURSOR ***
KO 1420 BLINKX=(BLINKX+1) MOD 20: IF BLINKX<10 THE
       N 147Ø ELSE 144Ø
PJ 1430 REM *** CURSOR OFF
                            ***
SH 1440 IF A(ROW, COLUMN) = 0 THEN CH$="." ELSE IF A
       (ROW, COLUMN) = 1 THEN CH$="#"
F 1450 GOTO 1480
SM 1460 REM *** CURSOR DN
                            ***
10 1470 IF CURS=-1 THEN CH$="-" ELSE IF CURS=0 TH
       EN CH$="$" ELSE IF CURS=1 THEN CH$="+"
EL 1480 LOCATE 3+ROW, 13+COLUMN: PRINT CH$:: RETURN
IC 149Ø REM *** REMOVE CURSOR
                               東東東
SN 1500 IF A(ROW, COLUMN) = THEN CH$="." ELSE IF A
       (ROW, COLUMN) = 1 THEN CH$="#"
OF 1510 LOCATE 3+ROW, 13+COLUMN: PRINT CH$;: RETURN
@ 1520 LOCATE 24,18:FRINT "wait";:FOR I=1 TO 20:
       LOCATE 3+1,14:PRINT STRING$(54,46);
EM 153Ø NEXT: ERASE A: DIM A(20,54): LOCATE 24,18: PR
       INT"
               ";:PUT(1,50),U%, PSET: RETURN
CD 154Ø REM *** SET CURSOR ***
N 1550 ROW=1:COLUMN=1:CURS=0
NF 1560 REM *** MAIN LOOP
                            東東東
EH 157Ø BLINKX=Ø:IF CURS=-1 THEN A(ROW,COLUMN)=Ø:
       PSET(COLUMN, ROW+49), Ø ELSE IF CURS=+1 THE
       N A(ROW, COLUMN) =1: PSET(COLUMN, ROH+49), 1
IL 1580 GOSUB 1420
P8 1590 A$=INKEY$:DEF SEG:POKE 104,0:IF LEN(A$)=0
        THEN 1580 ELSE IF LEN(A$)=1 THEN 1600 EL
       SE IF LEN(A$)=2 THEN 1720
8H 1600 CODE1=ASC(A$) AND &H5F
FE 1610 REM *** READ KEYS ***
OF 1620 IF CODE1=ASC("E") THEN 2040
PH 163Ø IF CODE1=ASC("M") THEN 2Ø5Ø
6F 164Ø IF CODE1=ASC("D") THEN 2Ø6Ø
JC 1650 IF CODE1=ASC("C") THEN 2080
@L 166Ø IF CODE1=ASC("S") THEN 21ØØ
60 1570 IF CODE1=ASC("Q") THEN GOSUB 2310:GOTO 19
PP 168Ø IF CODE1=ASC("I") THEN 171Ø
DE 169Ø GOTO 158Ø
KG 1700 REM ***
               INVERSE A PICTURE
UF 1710 GET (1,50) - (54,70), U%: PUT (1,50), U%, PRESET:
       GET(1,75)-(54,95),U%:GOTO 134Ø
EL 1720 IF ASC(A$)<>0 THEN 1570 ELSE CODE2=ASC(RI
```

GHT\$(A\$,1)):GOSUB 1500 HB 1730 REM \*\*\* READ ARROW KEYS

\*\*\*

```
IC 1740 IF CODE2=71 THEN 1840
PA 175Ø IF CODE2=73 THEN 187Ø
MM 176Ø IF CODE2=79 THEN 190Ø
KJ 1770 IF CODE2=81 THEN 1930
PI 1780 IF CODE2=72 THEN 1960
FN 1790 IF CODE2=75 THEN 1980
NL 1800 IF CODE2=77 THEN 2000
KP 1810 IF CODE2=80 THEN 2020
CA 1820 GOTO 1580
OK 1830 REM ***
                MOVE THE CURSOR
EJ 1840 IF ROW=1
               THEN ROW=21
FH 1850 IF COLUMN=1 THEN COLUMN=55
LE 1860 ROW=ROW-1:COLUMN=COLUMN-1:GOTO 1570
EC 187Ø IF ROW=1 THEN ROW=21
NN 1880 IF COLUMN=54 THEN COLUMN=0
ID 1890 ROW=ROW-1:COLUMN=COLUMN+1:GOTO 1570
LH 1900 IF ROW=20 THEN ROW=0
FN 1910 IF COLUMN=1 THEN COLUMN=55
JA 1920 ROW=ROW+1:COLUMN=COLUMN-1:60T0 1570
MA 1930 IF ROW=20 THEN ROW=0
LD 1940 IF COLUMN=54 THEN COLUMN=0
&P 1950 ROW=ROW+1:COLUMN=COLUMN+1:GOTO 1570
E8 1960 IF ROW=1 THEN ROW=21
M :970 ROW=ROW-1:GOTO 1570
SC 1980 IF COLUMN=1 THEN COLUMN=55
BN 1990 COLUMN=COLUMN-1:GOTO 1570
JE 2000 IF COLUMN=54 THEN COLUMN=0
JB 2010 COLUMN=COLUMN+1:GOTO 1570
LM 2020 IF ROW=20 THEN ROW=0
GF 2030 ROW=ROW+1:GOTO 1570
N 2040 CURS=-1:GOTO 1570
CJ 2060 CURS=+1:GOTO 1570
NG 2070 REM ***
                CLEAR THE PICTURE
BN 2080 GOSUB 1520:GOTO 1550
HE 2090 REM ***
                SAVE THE PICTURE
                                  ***
PD 2100 LOCATE 1,24:PRINT"WAIT...": ON B GOTO 2110
       ,2120,2130,2140,2150,2160,2170,2180,2190,
       2200, 2210, 2220, 2230, 2240, 2250, 2260, 2270, 2
       280,2290,2300
₩ 2110 GET(1,50)-(54,70),A%:GOSUB 2310:GOTO 190
LM 2120 GET(1,50)-(54,70), B%:GOSUB 2310:GOTO 190
₩ 2130 GET(1,50)-(54,70),C%:GOSUB 2310:GOTO 190
00 2140 GET(1,50)-(54,70), D%:GOSUB 2310:GOTO 190
Cl 2160 GET(1,50)-(54,70),F%:GOSUB 2310:GOTO 190
D 2170 GET(1,50)-(54,70),G%:GOSUB 2310:GOTO 190
FO 2180 GET(1,50)~(54,70),H%:GOSUB 2310:GOTO 190
HJ 2190 GET(1,50)-(54,70),IX:GOSUB 2310:GOTO 190
HI 2200 BET(1,50)-(54,70),J%:GOSUB 2310:GOTO 190
```

```
10 2210 GET(1,50)-(54,70), K%: GOSUB 2310: GOTO 190
KO 222Ø GET(1,5Ø)-(54,7Ø),L%:GOSUB 231Ø:GOTO 19Ø
HJ 2230 GET(1,50)-(54,70),M%:GOSUB 2310:GOTO 190
₩ 224Ø GET(1,5Ø)-(54,7Ø),N%:GOSUB 231Ø:GOTO 19Ø
PP 2250 GET(1,50)-(54,70),0%:GOSUB 2310:GOTO 190
BK 2260 GET(1,50)-(54,70),P%:GOSUB 2310:GOTO 190
CF 2270 GET(1,50)-(54,70),Q%:GOSUB 2310:GOTO 190
EA 2280 GET(1,50)~(54,70),R%:GOSUB 2310:GOTO 190
6L 229Ø GET(1,5Ø)-(54,7Ø),S%:GOSUB 231Ø:GOTO 19Ø
SK 2300 GET(1,50)-(54,70), T%: GCSUB 2310: GOTO 190
6K 231Ø CLS:PUT(1,1Ø),A%:PUT(61,1Ø),B%:PUT(121,1Ø
       ),C%:PUT(181,10),D%:PUT(241,10),E%:PUT(30
       1,10),F%:PUT(361,10),G%:PUT(421,10),H%:PU
       T(481,10), I%: PUT(541,10), J%
CA 2320 PUT(1,81), K%: PUT(61,81), L%: PUT(121,81), M%
       :PUT(181,81),N%:PUT(241,81),0%:PUT(301,81
       ),P%:PUT(361,81),Q%:PUT(421,81),R%:PUT(48
       1,81),5%:PUT(541,81),T%,PSET
JL 233Ø RETURN
8M 234Ø IF INKEY$="" THEN 234Ø ELSE 62Ø
PL 2350 REM *** INSERT A BLANK FICTURE
                                           ***
H 2360 LOCATE 18,1:PRINT"ARE YOU SURE?
       L MOVE 20 OFF THE END"
HC 2370 AS=INKEYS: IF AS="" THEN 2370 ELSE IF AS<>
       "Y" THEN LOCATE 18,1:PRINT STRING$ (50,32)
       :GOTO 190
FF 238Ø ON NUM GOTO 239Ø,240Ø,241Ø,242Ø,243Ø,244Ø
       ,2450,2460,2470,2480,2490,2500,2510,2520,
       2530, 2540, 2550, 2560, 2570, 2820
MH 239Ø GET(1,1Ø)-(54,3Ø),B%
LD 24ØØ GET(61,1Ø)-(114,3Ø),C%
F6 241Ø BET(121,1Ø)-(174,3Ø),D%
IJ 242Ø GET(181,1Ø)-(234,3Ø),E%
PN 243Ø GET(241,1Ø)-(294,3Ø),F%
LM 2440 GET (301, 10) - (354, 30), G%
NP 245Ø GET (361, 1Ø) - (414, 3Ø), H%
ED 246Ø GET (421, 1Ø) - (474, 3Ø), I%
% 247Ø GET(481,1Ø)-(534,3Ø),J%
OK 2480 GET (541, 10) - (594, 30), K%
PO 249Ø GET(1,81)~(54,1Ø1),L%
FA 2500 GET (61,81) - (114,101), M%
EK 251Ø GET(121,81)-(174,1Ø1),N%
60 252Ø GET(181,81)-(234,1Ø1),0%
ND 253Ø GET(241,81)-(294,1Ø1),P%
JD 254Ø GET(3Ø1,81)-(354,1Ø1),Q%
IH 255Ø GET (361,81) - (414,1Ø1),R%
DM 256Ø GET(421,81)-(474,1Ø1),S%
FA 257Ø GET (481,81) - (534,101), T%
```

```
AB 258Ø CLS: ON NUM GOTO 259Ø, 26ØØ, 261Ø, 262Ø, 263Ø,
       2640, 2650, 2660, 2670, 2680, 2690, 2700, 2710, 2
       720,2730,2740,2750,2760,2770,2780
LB 2590 GET (1,50) - (54,70), A%: GOSUB 2310: GOTO 190
LA 2600 GET (1,50) - (54,70), B%: GOSUB 2310: GOTO 190
NL 261Ø GET(1,5Ø)-(54,7Ø),C%:GOSUB 231Ø:GOTO 19Ø
06 2620 GET(1,50)-(54,70),D%:GOSUB 2310:GOTO 190
AB 2630 GET(1,50)-(54,70), E%: GOSUB 2310: GOTO 190
CM 2640 GET(1,50)-(54,70),F%:GOSUB 2310:GOTO 190
DH 2650 GET (1,50) - (54,70), G%: GOSUB 2310: GOTO 190
FC 2660 GET(1,50)-(54,70), H%: GOSUB 2310: GOTO 190
HN 2670 GET (1,50) - (54,70), I%: GOSUB 2310: GOTO 190
# 2680 GET(1,50)-(54,70), J%: GOSUB 2310: GOTO 190
KD 2690 GET(1,50)~(54,70),K%:GOSUB 2310:GOTO 190
KC 2700 GET(1,50)-(54,70),L%:GOSUB 2310:GOTO 190
MN 2710 GET(1,50)-(54,70), MX:GOSUB 2310:GOTO 190
81 2720 GET(1,50)-(54,70),N%:GOSUB 2310:GOTO 190
PD 2730 GET(1,50)-(54,70),-0%:60SUB 2310:60TO 190
80 274Ø GET(1,5Ø)-(54,7Ø),P%:GOSUB 231Ø:GOTO 19Ø
N 2750 GET(1,50)-(54,70),0%:GOSUB 2310:GOTO 190
EE 2760 GET(1,50)-(54,70),R%:GOSUB 2310:GOTO 190
6P 277Ø GET(1,5Ø)-(54,7Ø),S%:GOSUB 231Ø:GOTO 19Ø
IK 2780 GET(1,50)-(54,70),T%:GOSUB 2310:GOTO 190
EH 2790 REM *** DELETE A PICTURE
                                     ***
BM 2800 LOCATE 18,1:PRINT"ARE YOU SURE YOU WANT T
       O DELETE THIS NUMBER"
EF 2810 A$=INKEY$: IF A$="" THEN 2810 ELSE IF A$<>
       "Y" THEN LOCATE 18,1:PRINT STRING$ (50,32)
       :GOTO 19Ø
NH 2820 ON NUM GOTO 2830,2840,2850,2860,2870,2880
       , 2890, 2900, 2910, 2920, 2930, 2940, 2950, 2960,
       2970, 2980, 2990, 3000, 3010, 3020
JL 2830 GET (61,10) - (114,30), A%
DB 284Ø GET(121,10)~(174,30),B%
FE 285Ø GET(181,1Ø)-(234,3Ø),C%
MI 2860 GET(241,10)-(294,30),D%
IH 2870 GET (301,10) - (354,30),E%
ik 2880 GET(361,10)-(414,30),F%
CO 289Ø GET(421,1Ø)-(474,3Ø),G%
CF 2900 GET(481,10)-(534,30),H%
KJ 291Ø GET(541,1Ø)-(594,3Ø),I%
LB 292Ø GET(1,81)-(54,1Ø1),J%
DL 293Ø GET(61,81)~(114,1Ø1),K%
BD 294Ø GET(121,81)-(174,1Ø1),L%
DH 2950 GET(181,81)-(234,101),M%
LM 296Ø GET(241,81)-(294,101),N%
HM 297Ø GET(3Ø1,81)-(354,1Ø1),0%
JA 298Ø GET (361,81) - (414,1Ø1), F%
AF 2990 GET (421,81)-(474,101), Q%
OK 3000 GET (481,81) - (534,101),R%
```

```
#P 3010 GET (541,81)-(594,101),5%
IP 3Ø2Ø CLS:GET(1.1Ø)-(54.3Ø).T%:GOSUB 231Ø:GOTO
       190
#I 3030 REM ***
                 INVERSE ALL THE PICTURES
IN 3040 CLS:PUT(1,10),A%,PRESET:PUT(61,10),B%,PRE
       SET: PUT (121, 10), C%, PRESET: PUT (181, 10), D%,
       PRESET: PUT (241, 10), E%, PRESET: PUT (301, 10),
       F%, PRESET: PUT (361, 10), G%, PRESET: PUT (421, 1
       Ø), H%, PRESET: PUT (481, 10), I%, PRESET: PUT (54
       1,10), J%, PRESET
MM 3050 PUT(1,81), K%, PRESET: PUT(61,81), L%, PRESET:
       PUT(121,81), M%, PRESET: PUT(181,81), N%, PRES
       ET: PUT (241,81),0%, PRESET: PUT (3Ø1,81),P%,P
       RESET: PUT (361,81), Q%, PRESET: PUT (421,81), R
       %, PRESET: PUT (481,81), S%, PRESET: PUT (541,81
       ), T%, PRESET
E6 3060 GOTO 190
IF 3070 REM ***
                 MAKE A PROGRAM ***
BN 3080 LOCATE 18,1:PRINT"ARE YOU SURE YOU WANT T
       O MAKE THIS SET-UP INTO A PROGRAM?"
PJ 3090 A$=INKEY$: IF A$="" THEN 3090 ELSE IF A$="
       Y" THEN 3100 ELSE LOCATE 18,1:PRINT STRIN
       G$(66,32):GOTO 29Ø
NN 3100 G=30:0PEN "O",#1, "PRG.BAS"
PL 3110 PRINT #1,"10 CLS:KEY OFF:SCREEN 2:SP="+ST
       R$(SP)+":PL=1"
A 3120 IF STADEN THEN QQ=-1 ELSE QQ=1
EF 3130 AS="20 DIM ":FOR I=STA TO EN STEP QQ:IF I
       <>STA THEN A$=A$+","
KE 3140 ON I GOTO 3150,3160,3170,3180,3190,3200,3
       210,3220,3230,3240,3250,3260,3270,3280,32
       90,3300,3310,3320,3330,3340
M 315Ø A$=A$+"A%(144)":GOTO 335Ø
CM 316Ø A$=A$+"B%(144)":GOTO 335Ø
DM 317Ø A$=A$+"C%(144)":GOTO 335Ø
EM 318Ø A$=A$+"D%(144)":GOTO 335Ø
FM 319Ø A$=A$+"E%(144)":GOTO 335Ø
EA 3200 A$=A$+"F%(144)":GOTO 3350
FA 321Ø A$=A$+"G%(144)":GOTO 335Ø
64 322Ø A$=A$+"H%(144)":GOTO 335Ø
HA 323Ø A$=A$+"I%(144)":GOTO 335Ø
IA 3240 A$=A$+"J%(144)":GOTO 3350
JA 3250 A$=A$+"K%(144)":GDTO 3350
KA 326Ø A$=A$+"L%(144)":GOTO 335Ø
LA 327Ø A$=A$+"M%(144)":GOTO 335Ø
MA 328Ø A$=A$+"N%(144)":GOTO 335Ø
NA 329Ø A$=A$+"0%(144)":GOTO 335Ø
ME 3300 A$=A$+"P%(144)":GOTO 3350
NE 331Ø A$=A$+"Q%(144)":GOTO 335Ø
8E 332Ø A$=A$+"R%(144)":GOTO 335Ø
```

```
PE 3330 A$=A$+"5%(144)":GOTO 3350
AE 334Ø A$=A$+"T%(144)":GOTO 335Ø
AD 3350 NEXT
NI 3360 A$=A$+".U%(144)":PRINT #1.A$:PRINT #1."30
        GET(1,1)~(54,20),U%"
KD 3370 IF STA>EN THEN QQ=-1 ELSE QQ=1
16 3380 FOR P=STA TO EN STEP QQ
CE 3390 CLS: ON P GOTO 3400,3410,3420,3430,3440,34
       50,3460,3470,3480,3490,3500,3510,3520,353
       Ø,354Ø,355Ø,356Ø,357Ø,358Ø,359Ø
Pi 3400 PUT(1,50),A%:GOTO 3600
@ 3410 PUT(1,50), B%: GOTO 3600
80 3420 PUT(1,50), C%: GOTO 3600
CB 343Ø PUT(1,5Ø), D%: GOTO 36ØØ
DE 344Ø PUT(1,5Ø),E%:GOTO 36ØØ
EH 3450 PUT(1,50), F%: GOTO 3600
8K 346Ø PUT(1,5Ø),G%:GOTO 36ØØ
HN 3470 PUT (1,50), H%: GOTO 3600
IA 3480 PUT(1,50), IX:GOTO 3600
JD 3490 PUT(1,50),J%:GOTO 3600
※ 3500 PUT(1,50)、K%:GOTO 3600
KN 3510 PUT(1,50), L%: GOTO 3600
LA 3520 PUT(1,50), M%:GOTO 3600
#0 353Ø PUT(1,5Ø),N%:GOTO 36ØØ
M6 3540 PUT (1,50),0%:GOTO 3600
PJ 3550 PUT(1,50),P%:GOTO 3600
₩ 3560 PUT(1,50),Q%:GOTO 3600
$P 357Ø PUT(1,5Ø).R%:GOTO 36ØØ
CC 358Ø PUT(1,5Ø),S%:GOTO 36ØØ
OF 3590 PUT(1,50), T%:GOTO 3600
M 3600 FOR X=1 TO 54:G=G+10:A$=STR$(G):A$=RIGHT$
       (A$, LEN(A$)-1):FOR I=1 TO 20:IF POINT(X, I
       +49)=1 THEN A$=A$+":PSET(":B$=STR$(X):A$=
       A$+RIGHT$(B$,LEN(B$)-1):A$=A$+",":B$=STR$
       (I+49):A$=A$+RIGHT$(B$,LEN(B$)-1):A$=A$+"
FH 361Ø NEXT: IF LEN(A$)>6 THEN B$≃LEFT$(A$, LEN(ST
       R$(G))-1):A$=RIGHT$(A$,(LEN(A$)-LEN(B$))-
       1):A$=B$+" "+A$:PRINT #1,A$ ELSE G=G-10
AA 3620 NEXT
D 3630 G=G+10:A$=STR$(G):A$=RIGHT$(A$,LEN(A$)-1)
       :ON P GOTO 3640,3650,3660,3670,3680,3690,
       3700,3710,3720,3730,3740,3750,3760,3770,3
       780,3790,3800,3810,3820,3830
왜 3640 A$=A$+" GET(1,50)-(54,70),A%":GOTO 3840
CP 3650 A$=A$+" GET(1,50)-(54,70),B%":GOTO 3840
EC 3660 A$=A$+" GET(1,50)-(54,70),C%":GOTO 3840
&F 367Ø A$=A$+" GET(1,5Ø)-(54,7Ø),D%":GOTO 384Ø
J! 3680 A$=A$+" GET(1,50)-(54,70),E%":GOTO 3840
LL 3690 A$=A$+" GET(1,50) (54,70),FX":GOTO 3040
```

```
LC 3700 A$=A$+"
                GET (1,50) - (54,70), G%": GOTO 3840
NF 371Ø A$=A$+"
                GET (1,50) - (54,70), H%": GOTO 3840
01 372Ø A$=A$+"
                GET (1,50) - (54,70), T%": GOTO 3840
CL 373Ø A$=A$+" GET(1,5Ø)-(54,7Ø),J%":GOTO 384Ø
ED 3740 A$=A$+" GET(1,50)~(54,70),K%":GOTO 3840
69 375Ø A$=A$+" GET(1,5Ø)-(54,7Ø),L%":GOTO 384Ø
IE 3760 A$=A$+" GET(1,50)-(54,70),M%":GOTO 3840
KH 377Ø A$=A$+" GET(1,5Ø)-(54,7Ø),N%":GOTO 384Ø
NK 378Ø A$=A$+" GET(1,5Ø)-(54,7Ø),0%":GOTO 384Ø
PN 379Ø A$=A$+" GET(1,5Ø)-(54,7Ø),P%":GDTO 384Ø
PE 3800 A$=A$+" GET(1,50)-(54,70),0%":GOTO 3840
M 381Ø A$=A$+" GET(1,5Ø)-(54,7Ø),8%":G8TO 384Ø
EK 3820 A$=A$+" GET(1,50)-(54,70),5%":GOTO 3840
SN 383Ø A$=A$+" GET(1,5Ø)-(54,7Ø),T%":GOTO 384Ø
PN 3840 A$=A$+":CLS":PRINT #1,A$:NEXT
K8 3850 IF STA=EN THEN STN=1:EA=1:GOTO 3890
MK 3860 IF STADEN THEN STN=STA-EN: EA=1 ELSE EA=EN
       -STA: STN=1
ED 3870 IF EA+QQ=0 THEN EA=EA-QQ
OF 3880 G=G+10: R=G: A$=RIGHT$(STR$(G), LEN(STR$(G))
       -1) +" FOR I="+RIGHT$(STR$(STN), LEN(STR$(S
       TN))-1)+" TO"+STR$(EA+QQ)+" STEP"+STR$(QQ
       ):PRINT #1,A$:S=G+1Ø
M 389Ø A$=RIGHT$(STR$(G),LEN(STR$(G))-1)+" FOR P
       =1 TO": A$=A$+STR$(INT(FAST*4.2))+": NEXT:P
       L1=PL: IF PL+SP<1 THEN PL=570 ELSE IF PL+S
       P>58Ø THEN PL=1"
NJ 3900 G=G+10:PRINT #1,A$
FG 3910 A$=RIGHT$(STR$(G), LEN(STR$(G))-1)+" PL=PL
       +SP":PRINT #1,A$:G=G+1Ø
CK 3920 A$=RIGHT$(STR$(G), LEN(STR$(G))-1)+" ON I
       GOTO": X=G+10:A$=A$+STR$(X):FOR I=STA TO E
       N+(QQ*-1) STEP QQ: X=X+10: A$=A$+"."+RIGHT$
       (STR$(X), LEN(STR$(X))-1):NEXT:PRINT#1, A$
KL 3930 IF STADEN THEN Q=STA: W=EN ELSE IF ENDSTA
       THEN Q=EN: W=STA
01 3940 FOR P=W TO Q
PP 395Ø G=G+1Ø:A$=RIGHT$(STR$(G),LEN(STR$(G))-1):
       ON P GOTO 3960,3970,3980,3990,4000,4010,4
       020,4030,4040,4050,4060,4070,4080,4090,41
       00,4110,4120,4130,4140,4150
IE 3960 A$=A$+" PUT(PL1,52),U%,PSET:PUT(PL,52),A%
       :":GOTO 4160
LE 3970 A$=A$+" PUT(PL1,52),U%,PSET:PUT(PL,52),B%
       :":GOTO 416Ø
0E 398Ø A$=A$+" PUT(PL1,52),U%,PSET:PUT(PL,52),C%
       :":GOTO 4160
ME 3990 A$=A$+" PUT(PL1,52),U%,PSET:PUT(PL,52),D%
```

:":GOTO 4160

```
# 4000 A$=A$+" PUT(PL1,52),U%,PSET:PUT(PL,52),E%
       :":GOTO 416@
EF 4Ø1Ø A$=A$+" PUT(PL1,52),U%,PSET:PUT(PL,52),F%
       :":GOTO 416Ø
# 4020 A$=A$+" PUT(PL1,52),U%,PSET:PUT(PL,52),G%
       :":GOTO 416Ø
#F 4030 A$=A$+" PUT(PL1,52),U%,PSET:PUT(PL,52),H%
       :":GOTO 4160
# 4040 A$=A$+" PUT(PL1,52),U%,PSET:PUT(PL,52),I%
       :":GOTO 4160
AF 4050 A$=A$+" PUT(PL1,52),U%,PSET:PUT(PL,52),J%
       :":GOTO 416Ø
OF 4060 A$=A$+" PUT(PL1,52),U%,PSET:PUT(PL,52),K%
       :":GOTO 416Ø
6F 4070 A$=A$+" PUT(PL1,52),U%,PSET:PUT(PL,52),L%
       :":GOTO 4160
JF 4080 A$=A$+" PUT(PL1,52),U%,FSET:PUT(FL,52),M%
       :":GOTO 416Ø
#F 4090 A$=A$+" PUT(PL1,52),U%,PSET:PUT(PL,52),N%
       :":GOTO 416Ø
0J 4100 A$=A$+" PUT(PL1,52),U%,PSET:PUT(PL,52),0%
       :":GOTO 416Ø
BJ 4110 A$=A$+" PUT(PL1,52),U%,PSET:PUT(PL,52),P%
       :":GOTO 416Ø
87 4120 A$=A$+" PUT(PL1,52),U%,PSET:PUT(PL,52),Q%
       :":GOTO 416Ø
HJ 4130 A$=A$+" PUT(PL1,52),U%,PSET:PUT(PL,52),R%
       :":GOTO 416Ø
KJ 4140 A$=A$+" PUT(PL1,52),U%,PSET:PUT(PL,52),S%
       :":GOTO 4160
NJ 415Ø A$=A$+" PUT(PL1,52),U%,PSET:PUT(PL,52),T%
       :":GOTO 416Ø
00 4160 A$=A$+"GOTO"+STR$(X+10):PRINT #1, A$:NEXT:
       G=X+1Ø
LM 4170 A$=RIGHT$(STR$(G),LEN(STR$(G))-1):A$=A$+"
        NEXT: GOTO"+STR$ (R): PRINT#1, A$
NA 4180 CLOSE #1:CLS:PRINT"BEFORE YOU DO ANYTHING
        ELSE LOAD THE PROGRAM (PRG) AND THEN SAV
                        NAME YOU WANT": END
       E IT UNDER THE
```

# Million-Color Palette

John and Jeff Klein

It's amazing but true—with this stunning technique you can generate more than a million apparent color variations on a PCjr. You can even display 256 colors simultaneously. The effects are less dramatic on a PC, but it's still possible to generate many more than the standard 16 colors. The programs require an Enhanced Model PCjr or a PC with color/graphics adapter, plus a TV set or composite color monitor. The palette is more limited on an RGB monitor, but still impressive.

No longer is your PC or PCjr restricted to a palette of 16 colors and the inability to display them all in higher resolutions. Now you can choose to display 256 colors from a palette of over 1,000,000 colors in high resolution, and display an entire palette of 256 colors in medium resolution. And each color is distinct and solid.

The secret is a combination of a technique called *tile* painting and the trick of fooling a TV or composite monitor into displaying new solid colors. To understand how it works, let's examine the way graphics are stored, changed, and displayed on the IBM video screen.

A Byte of Pixels

Graphics images are stored differently in the computer's memory for each different graphics mode or screen. In its simplest form, the color of each *pixel*—the smallest controllable dot on the screen—is stored in a section of memory. This video memory is arranged by its location or coordinates on the screen. The image you see on the screen, therefore, is a copy of the contents of video memory. (Actually, screens are divided into several layers when stored in memory, but that's not important for this discussion; we're concerned with how the colors of pixels are represented in memory, not how each pixel is arranged.)

To figure out how many pixels can be represented in a byte of memory, remember that a byte is made up of eight bits, and a bit is the smallest unit of memory (a bit is either a zero or a one). Simply divide the amount of memory required for a certain screen mode by the number of pixels on the screen. The memory requirements for each screen mode are shown in Table 1.

Table 1. Screen Mode Memory Requirements

Screen Mode	Resolution	Number of Colors	Memory per Screen	Pixels/ Byte	Bits/ Pixel
1	$320 \times 200$	4	16K*	4	2
2	$640 \times 200$	2	16K	8	1
3	$160 \times 200$	16	16K	2	4
4	$320 \times 200$	4	16K	4	2
<sup>2</sup> 5	$320 \times 200$	16	32K	2	4
6	$640 \times 200$	4	32K	4	2

<sup>\*1</sup>K = 1024 bytes

Remember that RGB stands for the three primary colors of light: red, green, and blue. All colors can be made by mixing these three primary colors. That's why RGB monitors, color TVs, and composite color monitors have three electron guns inside their picture tubes, instead of the single gun found in black and white TVs and monochrome monitors. There is a red gun, a green gun, and a blue gun, all of which are controlled by the computer to produce color. If none of the guns is lighting a pixel, the pixel appears black.

Colors are represented in memory by arranging bits to denote which electron guns should be turned on or off when lighting the corresponding pixel. For instance, if a certain pixel is supposed to be blue, the group of bits representing that pixel in memory shows the blue gun is on and the others off. (A bit set to 1 means on, and 0 means off.) All the possible combinations of the three electron guns account for eight colors. To get eight more colors, the intensity, also called *luminance*, is varied by mixing a little white with the first eight colors. That's why the IBM PC and PCjr have a total of 16 color variations: two shades each of eight colors.

Table 2 shows how each of the 16 colors is represented. Remember that each bit turns an electron gun either on or off. Notice how many bits it takes to represent all the possible combinations. It takes four bits, or half a byte (sometimes called a *nybble*) to represent all 16 colors. So, all screen modes which use four bits to represent a pixel are 16-color modes. Only four color combinations are possible with two bits, and only two combinations are possible with one bit. That's why some screen modes can display only 4 or 2 colors at a time.

Table 2. Color Bits

Bits							
Luminance	Red	Green	Blue	Color			
0	0	0	0	Black			
0	0	0	1	Blue			
0	0	1	0	Green			
0	0	1	1	Cyan			
0	1	0	0	Red			
0	1	0	1	Magenta			
0	1	1	0	Brown			
0	1	1	1	Light gray			
1	0	0	0	Dark gray			
1	0	0	1	Light blue			
1	0	1	0	Light green			
1	0	1	1	Light cyan			
1	1	0	0	Pink			
1	1	0	1	Light magenta			
1	1	1	0	Yellow			
1	1	1	1	White			

The PCjr's PALETTE command can switch which colors are being displayed, but it can't add any more colors. You're still limited to the maximum number of colors for each screen mode.

# **Tile Painting**

Once you're familiar with how pixels are represented in video memory, the technique of *tile painting* is easier to understand. Tile painting uses the PAINT command found in PCjr Cartridge BASIC and IBM BASICA to fill the bytes of screen memory with certain patterns of ones and zeros. This pattern is programmable, and it represents what's displayed on the TV or monitor. Instead of painting with the actual color, you paint with the bit pattern of the color. By using bit patterns, you can

actually paint with more than one color around some specified border color.

# PAINT (x,y), CHR\$(bit pattern) + CHR\$(bit pattern) + ..., boundary color

The bit pattern consists of eight bits, so its decimal equivalent can range from 0 to 255 (integers only). The bit pattern must represent the colors of the pixels per byte of the screen mode you're using. This means four colors can be painted at a time in SCREEN 4 and 6, while only two colors can be painted at a time in SCREEN 3 and 5. The color patterns are put in memory next to each other as vertical lines on the screen. The following example paints SCREEN 1 with vertical bands of blue and green lines:

# 10 SCREEN 1:CLS 20 PAINT (1,1),CHR\$(102),3

The reason the lines are blue and green can be seen when the number 102 is expressed in binary, revealing the bit pattern:

### 102 = 01100110

Table 3 shows how decimal 102 is derived from this binary number. SCREEN 1 stores four pixels per byte, so the pattern works out to these colors:

Table 3. Converting Binary to Decimal

102

Value each			128	64	32	16	8	4	2	1	01 = 0001 = blue $10 = 0010 = $ green
Binar	ry		0	1	1	0	0	1	1	0	10 0010 green
128	*	0	=	0							
64	*	1	=	64							
32	*	1	=	32							
16	*	0	=	0							
8	*	0	=	0							
4 2	*	1	=	4							
2	*	1	=	2							
1	*	0	-	0							

Here's where things get tricky. If the computer is plugged into a color TV or composite color monitor (not an RGB monitor), you won't see the blue and green vertical lines which are supposed to be there. Instead, you'll see a solid bar of color that's sort of blue. And the blue is not one of the normal 16 colors available. It is a new color—one of the 256 shades that can be created this way on SCREEN 1 of the PCjr, and one of the 16 shades that can be created on SCREEN 1 of the PC.

What's happening here is something called *artifacting*. This effect takes advantage of the limited resolution of TVs and composite color monitors. When two very small pixels are placed next to each other on these screens, there isn't enough resolution to display them properly. As a result, the pixels tend to blend together and create a false color—an artifact color. The color wouldn't be visible if the screen had more resolution, which is why you usually need a TV or composite color monitor to observe this effect. RGB monitors have enough resolution to display the pixels as they're supposed to appear.

# **Creating New Colors**

If the binary pattern 10 01 10 01 is used in the above example instead of 01 10 01 10, the shade is slightly different—bluegreen-blue-green does not appear the same as green-bluegreen-blue on a color TV or or composite monitor. They mix differently to create an entirely new shade of blue-green.

The PC can mix a fewer number of colors than the PCjr for two reasons. The first is that the PC has only two graphics modes, SCREEN 1 and SCREEN 2. Tile painting produces only 16 colors in SCREEN 1 and five shades of gray in SCREEN 2. Still, these are more colors than are normally available in these modes. The second reason is that the PC does not have a PALETTE command as the PCjr does. The PC does have a second color palette in SCREEN 1, but the mixed colors look the same as the first palette on a color TV or composite monitor.

On SCREEN 6, available only in PCjr Cartridge BASIC, there are four pixels per byte. Because the pixels are very small ( $640 \times 400$  per screen), vertical bands of four different colors can be mixed to form shades of any color. In medium resolution,  $320 \times 200$ , vertical bands of two different colors

form new solid colors. Tile painting doesn't work in low resolution,  $160 \times 200$ , because the pixels are too large.

For a demonstration of how closely spaced vertical bands create new colors, enter and run Program 1 (for the PCjr only). Using the LINE command instead of PAINT, line 20 fills the first 40 columns of SCREEN 6 with purple bands on every line that's a multiple of four: 0, 4, 8, 12, and so on. Line 30 fills the next 40 columns with the same color on every vertical line that's a multiple of four plus one: 1, 5, 9, 13, and so on. Then the program fills the screen with lines of the other two colors available in SCREEN 6. The result, on a TV or composite monitor, is 12 different colors instead of the 4 you'd expect.

Adding up all the different combinations of 4 colors results in 256 shades, and all 256 can be displayed on the screen at the same time. When you take into account that the PALETTE command can change any of the four basic colors into any of the other 16 colors, there are 1,092,016 possible shades in high resolution.

Program 2 (again for the PCjr only) proves it can be done. This program displays 256 shades on the screen by drawing the vertical lines using only the first four colors. After painting all the shades, it randomly changes the palettes. If the colors selected by the PALETTE command were never repeated, it would take about an hour and a half to cycle through all one million colors.

# Colors in Other Modes

In SCREEN 5, there are 256 possible colors, as demonstrated by Program 3 (also for the PCjr only). In SCREEN 4 and SCREEN 1, which are the same resolution, only 4 basic colors are available, so tile painting lets us display up to 16 hues simultaneously. With the PALETTE command on the PCjr, you can select these 16 colors from 256 possibilities. Program 4 displays 16 shades, then uses the PALETTE command to get the rest. Vertical bands with 4 colors don't blend in this mode, so somehow bands of two must be painted. The secret is in line 40. Since there are four pixels per byte, the last half of the byte has to be reflected in the first half. This technique insures that only two colors are in each band of four. The first half is the same as the last half, so the first band of two will be the same as the last band of two. Program 4 will also work on the

PC, but without the PALETTE command (line 80), you are limited to only 16 colors.

Tile painting doesn't work correctly in SCREEN 2, high resolution with two colors, because this screen is always in black and white. However, you can get five shades of gray, as shown by Program 5 (for both the PC and PCjr). Solid lines form the brightest white. Lines separated by one line of black give the next-brightest white. Lines separated by two or three lines of black yield the next two shades. The middle gray can't be displayed when using the PAINT command, because it's not possible to create a bit pattern that represents two blacks and then a white. Table 4 shows which bit patterns generate the various shades of gray.

Table 4. Gray Scales in SCREEN 2

	Binary	<b>Decimal</b>	Hex	Shade
color 1 =	11111111	= 256 =	&HFF =	White
	01010101	= 85 =	&H55 =	Dull white
	(Not accessible)		-	Middle gray
	00010001	= 17 =	&H11 =	Dark gray
color 0 =	$0\ 0\ 0\ 0\ 0\ 0\ 0$	= 0 =	&H00 =	Black

Tile painting doesn't work at all in SCREEN 3 because the pixels are too large. To see a demo of tile painting in SCREEN 1 for the PC or PCjr, run Program 6. It fills the screen with circles, displaying up to 256 colors on the PCjr and 16 colors on the PC.

Program 7, for the PC and PCjr with an RGB monitor, demonstrates the usefulness of the many new colors in a fascinating experiment. It uses SCREEN 1 and tile painting, but in a different way from that seen above. Closely spaced vertical lines don't blend together on an RGB monitor, so the previous technique won't work. Instead, Program 7 uses the second part of the PAINT command. The first CHR\$(bit pattern) controls the horizontal line above the second CHR\$(bit pattern). Now the PAINT command can control the horizontal as well as the vertical lines, forming a checkerboard.

Although the checkerboard blends the lines together to create new colors, the colors aren't as solid as those produced by vertical lines on a TV or composite monitor. Indeed, the effect won't look very pretty on a TV or composite monitor; it's passable on an RGB.

Program 8 (for the PCjr only) employs the same technique as Program 7, but uses SCREEN 5 on the PCjr to create all 240 possible colors on the RGB monitor at once. The PALETTE command won't create any new shades here, because all 16 colors and their possible combinations are displayed.

Program 9 (for the PCjr only) is the same as the last two, but uses SCREEN 6 on the PCjr. It does a much better job of blending, although the colors still aren't perfectly solid. Ten shades are displayed at once, and the PALETTE command cycles through all 240 possible shades.

# **Painting Your Own Programs**

To use the new colors in your own programs, simply choose one of the following example programs which uses the same screen mode. Table 5 summarizes the programs and the number of color variations possible in each.

**Table 5. Program Descriptions** 

	Screen		Colors per		Display
Program	Mode	Max Colors	Screen	PC or PCjr	Device
Program 1	SCREEN 6	1,092,016	256	PCjr	TV or CC*
Program 2	SCREEN 6	1,092,016	256	PCjr	TV or CC
Program 3	SCREEN 5	256	256	PCjr	TV or CC
Program 4	SCREEN 1 or 4	256	16	PCjr	TV or CC
	SCREEN 1	16	16	PĆ	TV or CC
Program 5	SCREEN 2	5	5	PC/PCjr	TV or CC
Program 6	SCREEN 1	256	16	PCjr	TV or CC
		16	16	PĆ	TV or CC
Program 7	SCREEN 1 or 4	240	10	PCjr	RGB
	SCREEN 1	20	10	PC	RGB
Program 8	SCREEN 5	240	240	PCjr	RGB
Program 9	SCREEN 6	240	10	PCjr	RGB

<sup>\*</sup>CC = Composite color monitor

If you're programming on a PCjr, remove the lines that deal with changing the palette. You can change palettes on the PCjr in direct mode until most of the shades you want are on the screen. We suggest not changing the palettes in the 16-color modes, because the unchanged palette creates the widest variety of colors with the least amount of extra work.

In four-color modes, the screen displays 16 shades. Pick the color you want, then refer to Table 6 for the corresponding decimal and hexadecimal translations of the bit patterns required.

Table 6. Translations of Bit Patterns in Four-Color Modes

	TV or C	Composite	RGB			
<b>Shade Position</b>	Decima	l Hex	Decimal	Hex		
0	0	&H00	0 + 0	&H00 + &H00		
1	17	&H11	17 + 70	&H11 + &H44		
2	34	&H22	34 + 136	&H22 + &H88		
3	51	&H33	51 + 204	&H33 + &HCC		
4	70	&H44	70 + 17	&H44 + &H11		
5	85	&H55	85 + 85	& $H55 + &H55$		
6	102	&H66	102 + 153	&H66 + &H99		
7	119	&H77	119 + 221	&H77 + &HDD		
8	136	&H88	136 + 34	&H88 + &H22		
9	153	&H99	153 + 102	&H99 + &H66		
10	176	&HAA	176 + 176	&HAA+ &HAA		
11	187	&HBB	187 + 238	&HBB $+$ &HEE		
12	204	&HCC	204 + 51	&HCC + &H33		
13	221	&HDD	221 + 119	&HDD+ &H77		
14	238	&HEE	238 + 187	&HEE $+$ &HBB		
15	255	&HFF	255 + 255	&HFF + &HFF		

If you're using a 16-color mode with a TV or composite monitor, the screen displays 256 shades and the bit patterns can be figured as follows: First choose the color. Then, starting at zero at the upper-left corner of the screen, count in hex across the screen to the column with the color you want. Remember to count in hex (0 through 9, then A through F). Then, still working in hex, count the number of rows down to the color you want. These two numbers form the bit pattern of the chosen color. Use them as shown below:

# PAINT (x,y),CHR\$( &H row column),boundary

So if row = A, and column = 2, then

# PAINT (x,y), CHR\$(&HA2), boundary

If you're using an RGB monitor with a 16-color mode, choose which two of the 16 colors to make into the checkerboard. Then write each of their numbers in hex (0–F). Use these numbers as the bit pattern as shown below. Switching the first and second colors will create the checkerboard.

# PAINT (x,y),CHR\$&H 1st color 2nd color) + CHR\$(&H 2nd color 1st color),boundary

Thus if first color = B (light cyan) and second color = 2 (green), then

# PAINT (x,y), CHR\$(&HB2) + CHR\$(&H2B)

IBM boasts of only the checkerboard technique for shading colors. I find the other method more fascinating. Now you can enhance your screens with a new palette of bright, solid colors, which formerly were thought to be impossible on an IBM.

For error-free program entry, be sure to use "The Automatic Proofreader," Appendix B. Also, see Table 5 for a description of the programs.

# Program 1. PCjr

- IE 10 CLEAR,,,32768!:SCREEN 6:CLS:KEY OFF
- N 20 FOR X=0 TO 40 STEP 4:LINE (X,0)-(X,200),3:N EXT
- JH 3Ø FOR X=41 TO 8Ø STEP 4:LINE (X,Ø)-(X,2ØØ),3: NEXT
- IA 40 FOR X=82 TO 120 STEP 4:LINE (X,0)-(X,200),3
  :NEXT
- 6J 5Ø FOR X=123 TO 16Ø STEP 4:LINE (X,Ø)-(X,2ØØ), 3::NEXT
- FA 60 FOR X=160 TO 200 STEP 4:LINE (X,0)-(X,200), 1:NEXT
- HN 70 FOR X=201 TO 240 STEP 4:LINE (X,0)-(X,200), 1:NEXT
- PD 80 FOR X=242 TO 280 STEP 4:LINE (X,0)-(X,200), 1:NEXT
- NK 90 FOR X=283 TO 320 STEP 4:LINE (X,0)-(X,200), 1:NEXT
- 00 100 FOR X=320 TO 360 STEP 4:LINE (X,0)-(X,200)
  ,2:NEXT
- MN 110 FOR X=361 TO 400 STEP 4:LINE (X,0)-(X,200)
  ,2:NEXT
- OL 120 FOR X=402 TO 440 STEP 4:LINE (X,0)-(X,200)
  .2:NEXT
- 66 130 FOR X=443 TO 480 STEP 4:LINE (X,0)-(X,200)
  ,2:NEXT

# Program 2. PCjr

- IE 10 CLEAR,,,32768!: SCREEN 6: CLS: KEY OFF
- LO 20 RANDOMIZE TIMER: Z=-1:A=INT(640/16)
- HB 30 FOR Y=0 TO 15
- DE 40 FOR X=0 TO 15: Z=Z+1

- ED 50 LINE (X\*A,Y\*12.5)-(X\*A+A,Y\*12.5+12.5),3,B
  PC 60 IF Z<>0 THEN PAINT (X\*A+1,Y\*12.5+1),CHR\$(Z)
- ## 70 LINE (X\*A, Y\*12.5) (X\*A+A, Y\*12.5+12.5), Ø, B
- JI BØ NEXT X,Y
- M 90 PALETTE RND\*3, RND\*15:GOTO 90

### Program 3. PCjr

- HK 10 CLEAR,,,32768!:SCREEN 5:CLS:KEY OFF
- BB 20 RANDOMIZE TIMER: Z=-1: A=INT (320/16)
- HB 3Ø FOR Y=Ø TO 15
- OE 40 FOR X=0 TO 15: Z=Z+1
- ED 50 LINE (X\*A,Y\*12.5)-(X\*A+A,Y\*12.5+12.5),3,B
- PC 60 IF Z<>0 THEN PAINT (X\*A+1,Y\*12.5+1),CHR\$(Z)
- MH 70 LINE (X\*A,Y\*12.5)-(X\*A+A,Y\*12.5+12.5),0,B
- JI 80 NEXT X, Y
- KC 90 GOTO 90

# Program 4. PC/PCjr

- CA 10 SCREEN 1:CLS:KEY OFF:COLOR .0
- HH 20 RANDOMIZE VAL(RIGHT\$(TIME\$,2)):Z=-1:A=INT(3 20/16):Y=0
- 00 3Ø FOR X=Ø TO 15: Z=Z+1
- II 40 LINE (X\*A,0)-(X\*A+A,200),3,B
- KI 50 IF Z<>0 THEN PAINT (X\*A+1,1), CHR\$(Z+Z\*16),3
- CD 60 LINE (X\*A,0)-(X\*A+A,200),0,B
- M 70 NEXT X
- 6L 8Ø PALETTE RND\*3,RND\*15:GOTO 8Ø: Remove this line for PC

# Program 5. PC/PCjr

- OF 10 SCREEN 2,1:CLS:KEY OFF
- HJ 20 FOR X=1 TO 100:LINE (X,1)-(X,200),1:NEXT X
- 0H 3Ø FOR X=1Ø1 TO 2ØØ STEP 2:LINE (X,1)-(X,2ØØ), 1:NEXT X
- HN 40 FOR X=201 TO 300 STEP 3:LINE (X,1)-(X,200), 1:NEXT X
- KD 50 FOR X=301 TO 400 STEP 4:LINE (X,1)-(X,200), 1:NEXT X
- IE 60 GOTO 60

# Program 6. PC/PCjr

- CA 10 SCREEN 1:CLS:KEY OFF:COLOR ,0
- LO 20 RANDOMIZE VAL (RIGHT\$ (TIME\$, 2))
- 60 3Ø X=RND\*32Ø:Y=RND\*2ØØ:R=RND\*1Ø+1Ø:TILE=INT(RN D\*(15)+1)
- BH 4Ø CIRCLE (X,Y),R,3:PAINT (X,Y),CHR\$(TILE+TILE \*16),3:CIRCLE (X,Y),R,Ø
- LE 50 IF RND\*10>8 THEN Cr RND\*3+1,RND\*15: Remove this line for PC
- 6A 6Ø GOTO 2Ø

# Program 7. PC/PCjr

- CA 10 SCREEN 1:CLS:KEY OFF:COLOR .0
- CD 20 RANDOMIZE VAL(RIGHT\$(TIME\$,2)):Z=-1:A=INT(3 20/16):Y=0:C=0
- 09 3Ø FOR X=Ø TO 15: Z=Z+1
- M 4Ø LINE (X\*A,Ø)-(X\*A+A,2ØØ),3,B:Y=Z+Z\*16:Q=Y\*4 :R=INT(Q/256):Q=Q-R\*256+R
- KF 50 IF Z<>0 THEN PAINT (X\*A+1,1),CHR\$(Y)+CHR\$(Q
  ),3
- CD 60 LINE (X\*A,Ø)-(X\*A+A,200),Ø,B
- M 70 NEXT X
- FC 80 [r RND\*3,RND\*15:GOTO 80: Remove this line for PC

# Program 8. PCjr

- # 10 CLEAR,,,32768!:SCREEN 5:CLS:KEY OFF
- 88 20 RANDOMIZE TIMER: Z=-1: A=INT (320/16)
- HB 30 FOR Y=0 TO 15
- OE 40 FOR X=0 TO 15: Z=Z+1
- J0 50 LINE (X\*A,Y\*12.5)-(X\*A+A,Y\*12.5+12.5),3,B:Q
  =(Z)\*16:R=INT(Q/256):Q=Q-R\*256+R
- 00 60 IF Z<>00 THEN PAINT (X\*A+1,Y\*12.5+1),CHR\$(Z)
  +CHR\$(Q),3
- H 70 LINE (X\*A, Y\*12.5) (X\*A+A, Y\*12.5+12.5), Ø, B
- JI 80 NEXT X,Y
- KC 90 GOTO 90

# Program 9. PCjr

- IE 10 CLEAR, , , 32768!: SCREEN 6: CLS: KEY OFF
- 88 20 RANDOMIZE TIMER: Z=-1: A=INT (640/16): Y=0
- 00 3Ø FOR X=Ø TO 15: Z=Z+1
- NN 40 LINE (X\*A,0)-(X\*A+A,200),3,8:Y=Z+Z\*16:Q=Y\*4:R=INT(Q/256):Q=Q-R\*256+R
- KF 5Ø IF Z<>Ø THEN PAINT (X#A+1,1),CHR\$(Y)+CHR\$(Q
  ),3
- CD 60 LINE (X\*A,0)-(X\*A+A,200),0,B
- RH 70 NEXT X
- BE 80 PALETTE RND\*3, RND\*15: GOTO 80

# **Rotating Graphics**

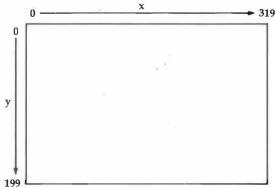
Michael A. Covington

Remember op art? Even if you aren't into mathematics or geometry, the formulas described in this article can help you create a fascinating variety of abstract figures. Both programs run in any version of IBM BASIC on the PC or PCjr.

Sometimes when you've figured out how to draw a certain pattern or shape on the screen of your PC or PCjr, you want to draw the same shape again in a different position. If all you want is to shift it up, down, or sideways, you need only to add or subtract constants from the x and y coordinates. But if you want to rotate or tilt it, you're up against something a good bit more complicated. This article shows how to perform rotations using three simple subroutines.

To begin with, you need to set up a standardized system of coordinates on the screen. The IBM identifies every screen location (*pixel*) with two coordinates (*x*, *y*); Figure 1 shows how they're numbered in medium-resolution graphics. The problem is that the computer's vertical and horizontal scales are not the same. Suppose, for instance, that you have a ten-inch screen, measured diagonally. Such a screen is six inches high and eight inches wide—which means the pixels are 40 to the inch horizontally but only 33.3 to the inch vertically. If you rotate a figure drawn with this coordinate system, the difference between horizontal and vertical scales change its shape.

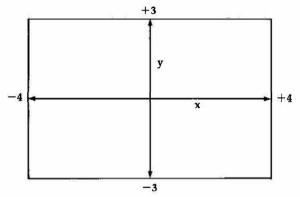
Figure 1. Medium-Resolution Screen Coordinates



# A New Coordinate System

The solution is to set up a coordinate system of your own with equal vertical and horizontal scales (Figure 2), and translate from this system to the IBM's system just before plotting. This is done in lines 10000–10070 of Program 1. The new system is called *standard rectilinear coordinates*.

Figure 2. Standardized Rectilinear Coordinates



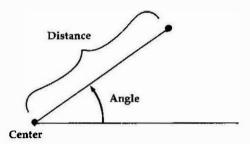
Because the IBM has an even number of pixels, the screen has no exact midpoint, and your coordinates don't have quite the range intended. The right margin represents x=+3.975 rather than +4, and the bottom margin represents y=-2.97 rather than -3. But this causes no serious problems in practice.

Now you have a coordinate system with which standard mathematical techniques for manipulating geometric figures will work correctly. The next step is to introduce *polar coordinates*. As Figure 3 shows, polar coordinates describe the position of any point in terms of its distance and direction from another point, referred to as the *center*. Rotation about the center is nothing more than changing the direction angle while keeping the distance the same.

The subroutines at lines 11000 and 12000 of Program 1 convert rectilinear coordinates to polar coordinates, and vice versa, taking point (XCEN,YCEN) as the center. [Remember that IBM BASIC variables are set to zero when the program starts running—as a result, if you don't give XCEN and YCEN any other values, the center is taken to be point (0,0) in the

middle of the screen.] The distances are measured in the same units as in the rectilinear coordinates you're converting from. The direction angles are measured in *radians*, where 6.28319 radians make a full circle. Angles greater than one full circle are perfectly all right. If you go around a complete circle, you end up where you started.

Figure 3. Polar Coordinates



# **Rotations and Spirals**

So, here's the general procedure for rotating a figure:

- **1.** Express the positions of the points in standard rectilinear coordinates.
- 2. Convert them to polar coordinates.
- 3. Add a constant to the direction angle of each point.
- 4. Convert back to standard rectilinear coordinates.
- 5. Convert to IBM screen coordinates and plot.

Program 1 puts this into practice by choosing triangles at random and creating multiple rotated images. Lines 280–340 define the triangle by choosing three points, each in standard rectilinear form, and immediately converting them to polar form. Then lines 380–390 decide how many steps the full circle of rotation will be divided into (the number of steps is displayed on the screen). Each time through the main loop, the program draws the triangle, then adds ROTANGL to the direction angle of each of the three points. The result is a spirograph-like repetitive pattern.

Program 2 shows another interesting trick you can do with polar coordinates. If you start with the distance and angle both as zero, and gradually increase both, thereby going around and around the circle while slowly moving out from

the center, you get a spiral. A circle results if the distance is constant. Making the distance vary in other ways results in a whole family of interesting geometric figures.

Program 2 continues to compute coordinates even after it has drawn lines off the screen, so press Break to stop it.

### **Program 1. Rotating Graphics**

```
@ 23Ø DIM A(3),D(3)
AH 240 RANDOMIZE VAL (RIGHT$ (TIME$, 2))
HP 250 '
EF 260 ' Choose 3 random points
CI 270 ' (in polar form)
6M 28Ø FOR I=1 TO 3
        X = 6 RND - 3
KL 290
JB 3ØØ
        Y = 6 RND - 3
DM 310
        GOSUB 11000
LB 320
        A(I) = ANGL
PP 330
       D(I) = DIST
NB 34Ø NEXT I
HA 35Ø '
JD 360 ' Choose number of steps
0A 37Ø ' of rotation
86 38Ø STEPS = 3+INT (6Ø*RND)
E0 390 ROTANGL = 6.28319 / STEPS
GH 4ØØ '
90 410 ' Main loop
LP 420 CLS : KEY OFF
66 43Ø SCREEN 1
NC 440 LOCATE 24,1 : PRINT STEPS;
10 45Ø FOR J = 1 TO STEPS
CB 46Ø
LD 470
        ' Draw the triangle
HN 48Ø
        ANGL = A(3)
        DIST = D(3)
EC 490
        GOSUB 12000 ' convert to rect.
HK 500
HD 510
        GOSUB 10000 ' convert to screen
NA 52Ø
        PSET(XPC, YPC) ' reference point
GN 53Ø
        FOR I=1 TO 3
KK 54Ø
          ANGL = A(I)
KJ 55Ø
          DIST = D(I)
DF 560
          GOSUB 12000 ' conv. to rect.
EI 570
          GOSUB 10000 ' conv. to screen
E0 58Ø
          LINE - (XPC, YPC)
JN 59Ø
        NEXT I
BH 600
        ' Apply the rotation
BL 61Ø
SH 620
        FOR I=1 TO 3
```

```
PE 630
          A(I) = A(I) + ROTANGL
IE 640
        NEXT I
CB 65Ø
OC 660 NEXT J
HH 670 '
DA 680 ' Pause, then go back
8% 69Ø FOR I=1 TO 1000 : NEXT I
FO 700 GOTO 260
HM 710 '
HD 720 '
IK 10000 ' Subroutine
D 10010 ' Converts standard rectilinear coordina
        tes X, Y
N 10020 ' to PC screen coordinates XPC, YPC
@N 10030 ' Range of X: -4 (left) to +3.975 (right
NI 10040 ' Range of Y: -2.97 (bottom) to +3 (top)
HJ 10050 XPC = 40*X + 160
CJ 10060 YPC = 100 - 33.33333*Y
IH 10070 RETURN
IN 11000 ' Subroutine
IF 11010 ' Converts standard rectilinear coordina
        tes X, Y
CN 11020 ' to polar coordinates DIST, ANGL
GN 11030 ' with respect to center XCEN, YCEN
LH 11040 XDIST = X-XCEN
NH 11050 YDIST = Y-YCEN
MI 11060 DIST = SQR(XDIST^2 + YDIST^2)
€ 11070 IF ABS(XDIST) < .000001 THEN XDIST=.0000
        Ø1
SF 11080 ANGL = ATN (YDIST/XDIST)
HM 11090 IF XDIST(0 THEN ANGL=ANGL+3.141593
HA 11100 RETURN
10 12000 ' Subroutine
KP 12010 ' Converts polar coordinates DIST, ANGL
DH 12020 ' to standard rectilinear coordinates X,
$P 12030 ' with respect to center XCEN, YCEN
JH 12040 X = DIST*COS(ANGL)
NH 12050 Y = DIST*SIN(ANGL)
IH 12060 RETURN
```

**Program 2. Spiral Graphics** 

```
BN 180 CLS
HP 190 SCREEN 1
GA 200 PRINT "Slope (number of units to"
NK 210 PRINT " move outward per rotation)";
```

```
IC 220 INPUT SLOPE
AE 23Ø CLS
HN 240 7
MN 25Ø DIST = Ø
EK 260 ANGL = 0
MP 270 GOSUB 12000
LJ 28Ø GOSUB 1ØØØØ
M 290 PSET(XPC, YPC) ' starting point
66 300 *
EC 310 ' Convert slope to units per radian
LI 320 SLOPE = SLOPE/6.28319
HM 330 "
AD 340 ' Main loop
BC 35Ø SPEED = .Ø5
                    ' Make higher to
FN 360
                    ' draw faster
JN 370 ANGL = ANGL + SPEED
MP 380 DIST = DIST + SPEED*SLOPE
ME 390 GOSUB 12000
KL 400 GOSUB 10000
NL 410 LINE - (XPC, YPC)
ED 420 GOTO 340
HN 43Ø *
DE 450 ' Remainder of program
80 460 ' the same as Program 1
00 470 ' except lines 11000-11100
88 480 ' are not needed.
IJ 49Ø ?
IK 10000 ' Subroutine
D 10010 ' Converts standard rectilinear coordina
        tes X, Y
JJ 10020 ' to PC screen coordinates XPC, YPC
@N 10030 ' Range of X: -4 (left) to +3.975 (right
NI 10040 ' Range of Y: -2.97 (bottom) to +3 (top)
HJ 10050 XPC = 40*X + 160
CJ 10060 YPC = 100 - 33.3333*Y
IH 10070 RETURN
10 12000 ' Subroutine
KP 12010 ' Converts polar coordinates DIST, ANGL
DH 12020 ' to standard rectilinear coordinates X,
6P 12Ø3Ø ' with respect to center XCEN, YCEN
JM 12040 X = DIST*COS(ANGL)
NH 12050 Y = DIST*SIN(ANGL)
IH 12060 RETURN
```

# First Steps in Three-Dimensional Graphics

Michael A. Covington

If you find two-dimensional computer graphics challenging, try your hand at simulating three-dimensional figures. The mathematics can quickly become intimidating, but the results are always fascinating. This article introduces the basic concepts without getting too technical and includes a couple of example programs. They require an IBM PCjr with Cartridge BASIC or a PC with BASICA and color/graphics adapter.

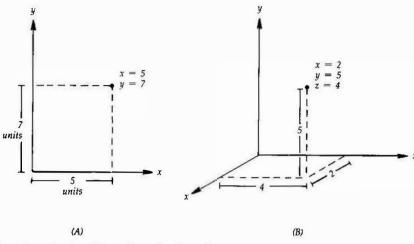
Computer graphics are normally a two-dimensional affair. The screen is flat, and directions are measured along two axes—the x-axis, which runs horizontally, and the y-axis, which runs vertically. But objects in the real world are three-dimensional, with depth as well as height and width. Let's explore some ways to bring flat-screen graphics closer to the real world by drawing three-dimensional objects as viewed from different directions.

Figure 1 shows how two-dimensional (x,y) and three-dimensional (x,y,z) coordinates work. In the three-dimensional system, the y-axis runs vertically, the z-axis runs horizontally, and the x-axis points directly toward you (the diagram shows it viewed from a slight angle).

A New Angle

The key to 3-D plotting is coordinate system translation. The idea is that three dimensions can be mapped onto two if you know the viewing angle. For example, if you're viewing the object straight on (so that the x-axis points directly toward you), you can ignore the front-to-back (x) dimension and treat the y and z dimensions as a two-dimensional system.





Two-dimensional and three-dimensional coordinate systems.

The equations in lines 3000–3250 of Program 1 deal with other viewing angles. They shift the entire 3-D coordinate system twice: first to allow you, the imaginary viewer, to move up or down, and then to let you move right or left. The result is a set of coordinates expressed in terms of a system in which the *x*-axis actually *does* point directly at the viewer, wherever you are, so that the *x* dimension can be ignored. The resulting projection is not true perspective—parallel lines don't converge to a vanishing point—but it's esthetically pleasing and easy to interpret.

Essentially, Program 1 lets you draw a picture in three dimensions. Each DATA statement contains four numbers: the x, y, and z coordinates of a point, and a number which tells whether you want to simply plot a point (0) or draw a line from a previously plotted point (1). The program begins by prompting you for the viewing angle, expressed in degrees up and to the right—relative to a straight front view. Then it draws your object on the screen as viewed from that angle. The point (0,0,0) is in the middle of the screen; you can go at least 2.5 units away from that point in any direction without going off the screen. (See Figures 2 through 6.)

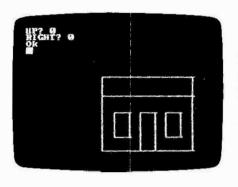


Figure 2
Program 1 lets you view a line drawing of a house from any elevation and clockwise rotation. This angle represents 0 degrees elevation and 0 degrees rotation—a

straight-on view.

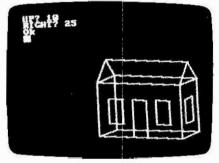


Figure 3
The same house, viewed from 10 degrees elevation and 25 degrees rotation.

You can also plot curves in three dimensions instead of drawing pictures with lines specified in DATA statements. To see an example, delete lines 500–2000 from Program 1 and replace them with the lines listed in Program 2. The result is a program that draws a conical helix viewable from any angle. (Figures 7 and 8.)

### The See-Through Problem

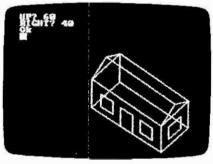
The main limitation of these programs is that they make no attempt to suppress the plotting of hidden surfaces. You see the back of the house as well as the front, and like some optical illusions, you can't always tell whether the helix is spiraling toward or away from you.

Hidden-surface elimination is one of the big headaches of computer-graphics programmers. Their programs must not only keep track of lines, but also of surfaces. The computer must know the difference between a solid house and a wire-frame model of a house. It takes large amounts of computer

Figure 4
20 degrees elevation, 45 degrees rotation.



Figure 5
60 degrees elevation and 40 degrees
rotation results in a bird's-eye view.



time to calculate which surfaces cover which points as seen from a particular viewing angle. That's why state-of-the-art 3-D graphics—as seen in film animation, for example—are done on very fast, very powerful supercomputers such as the Cray 1, Cray XMP, and Control Data Corporation CYBER series. (In fact, special effects in some films, such as the outer space scenes in *The Last Starfighter*, are not created with models as in the *Star Wars* movies, but with computer graphics on a Cray supercomputer.)

If you're interested in delving deeper into advanced graphics, two good books to read are Bruce A. Artwick's Applied Concepts in Microcomputer Graphics (Prentice-Hall, 1984) and Ian O. Angell's A Practical Introduction to Computer Graphics (Wiley, 1981). The book by Artwick (who's also the creator of the Flight Simulator software series) should be understandable by anyone who has mastered what the IBM BASIC manual has to say about graphics; Angell's book requires a good bit more mathematical knowledge, but gives a large number of handy formulas and algorithms.

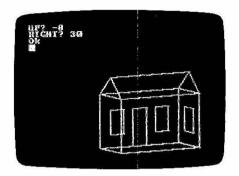


Figure 6
A negative elevation, —8 degrees, with 30 degrees rotation creates this "impossible" view, as if the house were seen from underground.

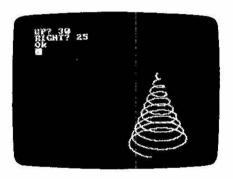


Figure 7
Substituting the lines in Program 2 for lines 500-2000 in Program 1 yields a 3-D image of a conical helix, seen here from 30 degrees elevation and 25 degrees rotation.



Figure 8
The helix viewed from 10 degrees elevation, 45 degrees rotation.

Program 1. 3-D Graphics

- HM 10 ' 3-D Graphics
- BJ 30 CLS : KEY OFF : SCREEN 1
- LH 40 PRINT "This program displays a predefined"
- EL 50 PRINT "3-dimensional shape as viewed from"
- OF 60 PRINT "different angles."
- FJ 7Ø PRINT
- AG 80 PRINT "The program asks for two numbers."
- ll 90 PRINT "'Right' is the number of degrees you
- JI 100 PRINT "want to move to the right, relative
- # 110 PRINT "to the front straight-on position. It"
- EA 120 PRINT "can be negative if you want to move
- OA 130 PRINT "to the left."
- JI 140 PRINT
- 8 150 PRINT "Similarly, 'Up' is the number of"
- LP 160 PRINT "degrees you want to move up."
- JO 170 PRINT
- 00 180 WHILE INKEY\$<>"" : WEND ' Clear buffer
- ON 185 PRINT "Be sure CAPS LOCK is on."
- LE 190 PRINT "Press spacebar to continue..."
- HH 200 WHILE INKEY\$="": WEND ' Wait for key
- HP 210 CLS: RESTORE
- AA 220 LOCATE 1,1:INPUT "UP";U\$:IF LEN(U\$)>4 THEN LOCATE 1,1:PRINT SPACE\$(10):GOTO 220
- KF 23Ø IF LEN(U\$)=4 AND (ASC(MID\$(U\$,1,1))<>45 AN
  D ASC(MID\$(U\$,1,1))<>43) THEN LOCATE 1,1:P
  RINT SPACE\$(10):GOTO 220
- PL 240 LOCATE 1,28:INPUT "RIGHT";R\$:IF LEN(R\$)>4 THEN LOCATE 1,28:PRINT SPACE\$(13):GOTO 240
- EK 250 IF LEN(R\$)=4 AND (ASC(MID\$(R\$,1,1))<>45 AND ASC(MID\$(R\$,1,1))<>43) THEN LOCATE 1,28: PRINT SPACE\$(13):60TO 240
- CG 260 U=VAL(U\$):R=VAL(R\$)
- 00 500 READ X,Y,Z,LINEMODE
- KP 510 IF X=999 THEN 2010
- PB 52Ø GOSUB 4ØØØ
- N 53Ø GOTO 5ØØ
- D6 540 ' Each of the following
- CF 550 ' DATA statements speci-
- AF 560 ' fies a point, plus a
- NL 570 ' code: 0 to move to the
- FB 580 ' point, 1 to draw a
- BK 590 ' line there from the
- IB 600 ' previously mentioned
- NE 610 ' point.

```
HN 620 "
FD 1900 ' front of house
E6 1010 DATA 0.7,-1.5,-1.5,0
NC 1020 DATA 0.7,0.5,-1.5,1
JN 1030 DATA 0.7, 0.5, 1.5, 1
PD 1040 DATA 0.7,-1.5,1.5,1
HL 1050 DATA 0.7,-1.5,-1.5,1
DP 1060 ' end of house
DE 1070 DATA 0.7,-1.5,1.5,0
JN 1080 DATA -0.7,-1.5,1.5,1
₽ 1090 DATA -0.7,0.5,1.5,1
16 1100 DATA 0.7,0.5,1.5,1
IP 1110 ' roof
MB 1120 DATA 0,1.2,1.5,1
PP 1130 DATA -0.7, 0.5, 1.5, 1
LC 1140 DATA 0,1.2,1.5,0
MI 1150 DATA 0, 1.2, -1.5,1
QA 1160 DATA 0.7, 0.5, -1.5, 1
NI 1170 ' back of roof
LL 1180 DATA 0,1.2,-1.5,0
HG 1190 DATA -0.7, 0.5, -1.5, 1
PI 1200 DATA -0.7, 0.5, 1.5, 1
MA 1210 DATA 0,1.2,1.5,1
DB 1220 ' remainder of back
HJ 1230 DATA -0.7,-1.5,1.5,0
CH 1240 DATA -0.7,-1.5,-1.5,1
#P 1250 DATA 0.7,-1.5,-1.5,1
80 1260 DATA -0.7,-1.5,-1.5,0
HC 1270 DATA -0.7,0.5,-1.5,1
PI 1280 DATA 0.7, 0.5, -1.5, 1
M 1290 ' front door
ED 1300 DATA 0.7,-1.5,-0.25,0
BI 1310 DATA 0.7,-0.1,-0.25,1
6F 132Ø DATA Ø.7, -Ø.1, Ø.25, 1
MA 1330 DATA 0.7,-1.5,0.25,1
OH 1340 ' windows
KE 1350 DATA 0.7,-0.1,-1.1,0
CO 1360 DATA 0.7,-0.1,-0.6,1
CN 1370 DATA 0.7,-1,-0.6,1
NK 1380 DATA 0.7,-1,-1.1,1
NJ 1390 DATA 0.7,-0.1,-1.1,1
CH 1400 DATA 0.7,-0.1,1.1,0
KM 1410 DATA 0.7,-0.1,0.6,1
PF 1420 DATA 0.7,-1,0.6,1
K6 143Ø DATA Ø.7,-1,1.1,1
FL 1440 DATA 0.7,-0.1,1.1,1
6K 145Ø DATA Ø.3,-Ø.1,1.5,Ø
NL 1460 DATA 0.3,-1,1.5,1
PF 1470 DATA -0.3,-1,1.5,1
AA 1480 DATA -0.3, -0.1,1.5,1
```

```
10 1490 DATA 0.3,-0.1,1.5,1
HK 2000 DATA 999,999,999,999
DH 2010 LOCATE 25,1: PRINT "Do you wish to draw ag
       ain (Y/N)?":
8 2020 Z$=INKEY$:IF Z$="" OR (Z$<>"Y" AND Z$<>"N
       ") THEN 2020
AB 2030 IF Z$="Y" THEN 210
IN 2040 PRINT CHR$ (11): END
PF 3000 ' Subroutine
00 3010 ' Given 3-dimensional coordinates X,Y,Z,
ID 3020 ' with X toward viewer, Y up, and Z to ri
       ght,
NP 3Ø3Ø ' performs transformation to allow for a
N 3040 ' viewing position U degrees above X-Z pl
       ane
JG 3050 ' and R degrees to right of X-Y plane,
KK 3060 ' then determines XPC, YPC for plotting
Li 3070 ' on screen
IF 3Ø8Ø
MW 3Ø9Ø ' Step 1: Allow for R
00 31000 R1=R/(45/ATN(1)) ' Conv to radians
AE 3110 X1 = X*COS(R1) + Z*SIN(R1)
0A \ 3120 \ Y1 = Y
LB 3130 Z1 = -X*SIN(R1) + Z*COS(R1)
BL 3140 ' Step 2: Allow for U
JP 3150 U1=U/(45/ATN(1)) ' conv to radians
CC 3160 X2 = X1*COS(U1) + Y1*SIN(U1)
DI 3170 Y2 = -X1*SIN(U1) + Y1*COS(U1)
K6 \ 3180 \ Z2 = Z1
FA 3190 ' Step 3: Put onto flat plane
0 3200 \text{ YR} = Y2 : XR = Z2
IC 3210 ' Step 4: Convert to PC's scale
LI 3220 YPC = 100 - 33.3333*YR
C6 323Ø XPC = 4Ø*XR + 16Ø
IN 3240 *
JA 3250 RETURN
% 4000 ' Subroutine
BB 4010 ' Given a point X,Y,Z in the 3-D coordina
       tes
JA 4020 ' described above, locates it on the scre
       en and plots a
PC 4030 ' point there if LINEMODE=0, or draws a l
       ine to it
LA 4040 ' from the previously plotted point if LI
       NEMODE <> Ø.
IN 4050 7
IJ 4060 GOSUB 3000 ' find XPC, YPC
88 4070 IF LINEMODE THEN LINE - (XPC, YPC) ELSE PSE
       T(XPC, YPC)
J6 4Ø8Ø RETURN
```

### Program 2. Conical Helix

- HN 10 ' 3-D Graphics
- BJ 30 CLS : KEY OFF : SCREEN 1
- LH 40 PRINT "This program displays a predefined"
- £1 50 PRINT "3-dimensional shape as viewed from"
- DF 60 PRINT "different angles."
- FJ 7Ø PRINT
- AG 80 PRINT "The program asks for two numbers."
- ll 90 PRINT "'Right' is the number of degrees you
- JI 100 PRINT "want to move to the right, relative
- AL 110 PRINT "to the front straight-on position. It"
- EA 120 PRINT "can be negative if you want to move
- DA 130 PRINT "to the left."
- JI 140 PRINT
- SP 150 PRINT "Similarly, 'Up' is the number of"
- LP 160 PRINT "degrees you want to move up."
- J0 170 PRINT
- CO 180 WHILE INKEY\$<>"" : WEND ' Clear buffer
- OM 185 PRINT "Be sure CAPS LOCK is on."
- LE 190 PRINT "Press spacebar to continue..."
- ## 200 WHILE INKEYS="": WEND ' Wait for key
- # 210 CLS: RESTORE
- AA 220 LOCATE 1,1:INPUT "UP"; U\$:IF LEN(U\$)>4 THEN LOCATE 1,1:PRINT SPACE\$(10):GOTO 220
- KF 23Ø IF LEN(U\$)=4 AND (ASC(MID\$(U\$,1,1))<>45 AN
  D ASC(MID\$(U\$,1,1))<>43) THEN LOCATE 1,1:P
  RINT SPACE\$(10):GOTO 220
- PL 240 LOCATE 1,28:INPUT "RIGHT";R\$:IF LEN(R\$)>4
  THEN LOCATE 1,28:PRINT SPACE\$(13):GOTO 240
- EK 250 IF LEN(R\$)=4 AND (ASC(MID\$(R\$,1,1))<>45 AND ASC(MID\$(R\$,1,1))<>43) THEN LOCATE 1,28: PRINT SPACE\$(13):GOTO 240
- CG 260 U=VAL(U\$): R=VAL(R\$)
- OF 1000 ' Draw a conical helix
- IO 1010 ' (Delete lines
- CB 1020 ' 500-2000 in Program 1
- KN 1030 ' and replace them
- IG 1040 ' with these)
- 6K 1050 X=SIN(-30):Y=-1.5:Z=COS(-30)
- BN 1060 LINEMODE=0:GOSUB 4000
- IH 1070 LINEMODE=1
- IE 1080 FOR T=-30 TO 30 STEP .1
- GL 1090 F=ABS((30-T)/60)
- JS 1100 X=SIN(T) \*F
- JC 1110 Y=T/20

```
JA 1120 Z=COS(T) *F
EI 113Ø GOSUB 4ØØØ
N6 1140 NEXT T
OH 2010 LOCATE 25,1:PRINT "Do you wish to draw ag
       ain (Y/N)?";
61 2020 Z$=INKEY$: IF Z$="" OR (Z$<>"Y" AND Z$<>"N
       ") THEN 2020
AB 2030 IF Z$="Y" THEN 210
PN 2040 PRINT CHR$(11):END
Pf 3000 ' Subroutine
© 3010 ' Given 3-dimensional coordinates X,Y,Z,
ID 3020 ' with X toward viewer, Y up, and Z to ri
# 3030 ' performs transformation to allow for a
M 3040 ' viewing position U degrees above X-Z pl
       ane
16 3050 ' and R degrees to right of X-Y plane,
KK 3060 ' then determines XPC, YPC for plotting
li 3070 ' on screen
IF 3Ø8Ø *
MN 3090 'Step 1: Allow for R
DD 3100 R1=R/(45/ATN(1)) ' Conv to radians
AE 3110 X1 = X*COS(R1) + Z*SIN(R1)
DA \ 3120 \ Y1 = Y
L8 313Ø Z1 = -X*SIN(R1) + Z*COS(R1)
BL 3140 ' Step 2: Allow for U
JP 3150 U1=U/(45/ATN(1)) ' conv to radians
CC 3160 X2 = X1*COS(U1) + Y1*SIN(U1)
013170 Y2 = -X1*SIN(U1) + Y1*COS(U1)
KG 318Ø Z2 = Z1
FA 3190 ' Step 3: Put onto flat plane
00\ 3200\ YR = Y2 : XR = Z2
IC 3210 ' Step 4: Convert to PC's scale
LI 322Ø YPC = 1ØØ - 33.3333≭YR
C6 323Ø XPC = 4Ø*XR + 16Ø
IN 3240 '
JA 3250 RETURN
PG 4000 ' Subroutine
BB 4010 ' Given a point X,Y,Z in the 3-D coordina
       tes
JA 4020 ' described above, locates it on the scre
       en and plots a
PC 4030 ' point there if LINEMODE=0, or draws a 1
       ine to it
LA 4040 ' from the previously plotted point if LI
       NEMODE <> Ø.
IN 4Ø5Ø *
IJ 4060 GOSUB 3000 ' find XPC, YPC
EB 4070 IF LINEMODE THEN LINE - (XPC, YPC) ELSE PSE
       T(XPC, YPC)
JE 4080 RETURN
```

### **Fractal Graphics**

Paul W. Carlson

One of the hottest topics in mathematics these days is fractals—fractional dimensions. Fractals are being used for everything from simulating random plant growth to generating realistic planetary landscapes for science-fiction films and arcade games. This article, adapted from "Apple Fractals" in the September 1985 issue of COMPUTE! magazine, introduces the fascinating world of fractals with three programs that work on any IBM PCjr or PC with color/graphics adapter.

The term *fractal* was coined by Benoit Mandelbrot, a pioneer in their study, to denote curves or surfaces having *fractional dimension*. The concept of fractional dimension can be illustrated as follows: A straight curve (a line) is one-dimensional, having only length. However, if the curve is infinitely long and curves about in such a manner as to completely fill an area of the plane containing it, the curve could be considered two-dimensional. A curve partially filling an area would have a fractional dimension between one and two.

Many types of fractals are *self-similar*, which means that all portions of the fractal resemble each other. Self-similarity occurs whenever the whole is an expansion of some basic building block. In the language of fractals, this basic building block is called the *generator*. The generator in the accompanying programs consists of a number of connected line segments. The curves that the programs plot are the result of starting with the generator and then repeatedly replacing each line segment with the whole generator according to a defined rule. Theoretically, these replacement cycles would continue indefinitely. In practice, the screen resolution limits the number of cycles.

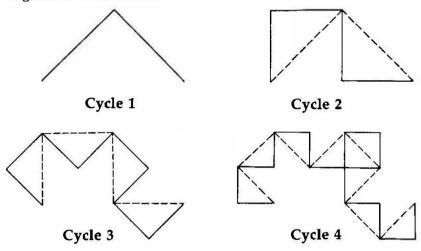
The programs illustrate two types of fractal curves. The curves generated by Programs 1 and 2 are *self-contacting*, while the curve generated by Program 3 is *self-avoiding*.

A self-contacting curve touches itself but does not cross itself. A self-avoiding curve never actually touches itself, although it may appear to because of the limited screen resolution.

The Dragon Sweep

Program 1 plots what Mandelbrot refers to as a *dragon sweep*. It demonstrates in a step-by-step fashion how a fractal curve is filled. The generator consists of two line segments of equal length forming a right angle. During each replacement cycle, the generator is substituted for each segment on alternating sides of the segments, that is, to the left of the first segment, to the right of the second segment, and so on. Figure 1 shows the first few cycles of substitution. The program is written in BASIC so the plotting is slow enough to let you see the development of the curve.

Figure 1. Substitution



The program prompts you to enter an even number of cycles (for reasons of efficiency and screen resolution, only even numbers of cycles are plotted). When a plot is complete, pressing any key clears the screen and returns you to the prompt. I recommend starting with 2 cycles, then 4, 6, and so on. It takes 14 cycles to completely fill in the "dragon," but since this requires almost two hours, you will probably want to quit after about 10 cycles. You can see the complete dragon by running Program 2, which always plots the dragon first in less than 30 seconds.

Since it's not at all obvious how the program works, here's a brief explanation. NC is the number of cycles; C is the

cycle number; SN is an array of segment numbers indexed by cycle number; L is the segment length; D is the segment direction, numbered clockwise from the positive x direction; and X and Y are the high-resolution screen coordinates.

Line(s)	Function
100-140	Get number of cycles from user.
150	Computes segment length.
160	Sets starting coordinates.
170	Sets segment numbers for all cycles to the first segment.
180 - 220	Find the direction of the segment in the last cycle by ro-
	tating the segment in each cycle that will contain the seg- ment in the last cycle.
230-260	
270–290	
300–320	If the segment number for cycle zero is still zero, do the next segment; otherwise, the program's done.

### **Eight Thousand Dragons**

Program 2 plots more than 8,000 different dragons. It does this by randomly determining on which side of the first segment the generator will be substituted for all cycles after the first. The generator is always substituted to the left of the first segment in the first cycle to avoid plotting off the screen. Other than the randomization, this program uses the same logic as Program 1. The main part of this program is written in machine language to reduce the time required to plot a completely filled-in dragon from about two hours to less than half a minute.

All the dragons are plotted after 14 cycles of substitution. All have exactly the same area, which equals half the square of the distance between the first and last points plotted. All the dragons begin and end at the same points.

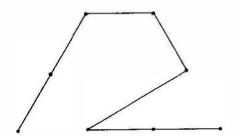
When a plot is complete, press the space bar to plot another dragon, or press the Q key to quit.

### **Snowflakes**

Program 3 plots what Mandelbrot refers to as a *snowflake* sweep. The generator, shown in Figure 2, was discovered by Mandelbrot. The segments are numbered zero through six, starting at the right. The program is basically the same as Pro-

gram 1. The variables NC, C, SN, D, X, and Y represent the same values except that the direction D is numbered counterclockwise from the negative x direction. For each segment, the following table gives the value of RD (relative direction), LN (length factor), and SD (flags indicating upon which side of the segment the generator is to be placed).

Figure 2. Snowflake Generator



Program	3 V	alues
---------	-----	-------

Segment Number SN	Relative Direction RD	Length Factor LN	Side Flag SD
0	0	1/3	0
1	0	1/3	1
2	7	$\sqrt{1/3}$	1
3	10	1/3	0
4	0	1/3	0
5	2	1/3	0
6	2	1/3	1

Here's how Program 3 is organized:

	-
Line(s)	Function
20	Reads values of SD and RD. Computes LN values.
30-50	Compute delta x and delta y factors for each direction.
60-100	Get number of cycles from user.
120	Sets starting coordinates.
130	Sets the segment numbers for all cycles to the first
	segment.
140 - 170	Find the direction of the segment in the last cycle.
180 - 190	Compute the coordinates of the end of the segment, plot
	the segment, and update the segment numbers for each
	cycle.
200-220	Same as lines 300-320 in Program 1

Like Program 1, pressing any key when a plot is complete clears the screen and brings another prompt.

### **Experiment**

I hope these programs encourage you to look further into the fascinating world of fractals. Don't be afraid to experiment with the programs—try modifying the shape of the generator in Program 3, for example. Better yet, design your own generator.

These programs just begin to explore the possibilities of fractal computer graphics. There is another whole class of fractals, those generated by functions of complex variables. And then there are three-dimensional fractals. And then...

#### Program 1. The Dragon Sweep

```
MH 90 DIM SN(14): KEY OFF
MN 100 CLS: SCREEN Ø
PI 110 PRINT"ENTER AN EVEN NO. OF CYCLES (2 TO 14
      ) "
                       OR ENTER A ZERO TO QUIT: "
DD 120 INPUT"
      ; NC
EH 130 IF NC = 0 THEN KEY ON: END
HO 140 IF NO MOD 2 = 1 OR NO < 2 OR NO > 14 THEN
DL 150 L=128:FOR C=2 TO NC STEP 2:L=L/2:NEXT
AE 160 X=192: Y=133: CLS: SCREEN 2: PSET (X,Y),1
DL 170 FOR C=0 TO NC:SN(C)=0:NEXT
KJ 180 D=0:FOR C=1 TO NC: IF SN(C-1)=SN(C) THEN D=
      D-1:60TO 200
6C 19Ø D=D+1
₩ 200 IF D=-1 THEN D=7
MI 210 IF D=8 THEN D=0
NC 22Ø NEXT
0A 23Ø IF D=Ø THEN X=X+L+L:GOTO 27Ø
EM 24Ø IF D=2 THEN Y=Y+L:GOTO 27Ø
6A 25Ø IF D=4 THEN X=X-L-L:GQTO 27Ø
8 260 Y=Y-L
PL 270 LINE -(X,Y), 1:SN(NC)=SN(NC)+1
@ 280 FOR C=NC TO 1 STEP -1: IF SN(C) <>2 THEN 300
OH 290 SN(C)=0:SN(C-1)=SN(C-1)+1:NEXT
GA 300 IF SN(0)=0 THEN 180
MM 31Ø IF INKEY$="" THEN 31Ø
AC 32Ø GOTO 1ØØ
```

#### **Program 2. Eight Thousand Dragons**

```
PP 100 DEF SEG: CLEAR, &H3FF0: N=&H4000
LL 110 READ AS: IF AS="/" THEN 130
LI 120 POKE N, VAL ("&H"+A$): N=N+1: GOTO 110
II 130 N=&H440F:FOR K=1 TO 15:POKE N, Ø:N=N+1:NEXT
MH 140 POKE &H4425,0
IF 150 N=&H4000: CALL N: POKE &H4425, 1
EA 160 AS=INKEYS: IF AS="" THEN 160
M 170 IF A$=" " THEN 150
ID 180 IF A$<>"Q" AND A$<>"q" THEN 160
IH 190 SCREEN Ø: CLS: KEY ON: END
EI 1000 DATA 1E, 0E, 1F, B8, 05, 00, CD, 10, 80, 3E
DN 1010 DATA 25,44,00,75,08,84,00,CD,1A,89
NF 1020 DATA 16,23,44,EB,31,90,BE,02,00,B9
OE 1030 DATA 08,00,A1,23,44,33,D2,A9,02,00
JE 1040 DATA 74,02,B2,01,A9,04,00,74,02,B6
NG 1050 DATA 01,32,D6,D0,EA,D1,D8,E2,E8,A3
N 1060 DATA 23,44,24,01,88,84,0F,44,46,83
FH 1070 DATA FE, 0F, 75, D3, B8, 00, 06, 33, C9, BA
LI 1080 DATA 4F, 18, 32, FF, CD, 10, B9, 0F, 00, 33
NJ 1090 DATA F6,C6,84,00,44,00,46,E2,F8,C7
JD 1100 DATA 06,1E,44,60,00,C7,06,20,44,84
₩ 1110 DATA 00,B8,01,0C,88,0E,1E,44,8B,16
№ 1120 DATA 20,44,CD,10,C6,06,22,44,00,B9
LO 1130 DATA 0E,00,33,FF,BE,01,00,8A,A5,0F
KC 114Ø DATA 44,8Ø,FC,ØØ,75,18,FE,Ø6,22,44
CL 1150 DATA 8A,85,00,44,3A,84,00,44,75,20
L6 1160 DATA FE, ØE, 22, 44, FE, ØE, 22, 44, EB, 16
00 1170 DATA FE, ØE, 22, 44, 8A, 85, ØØ, 44, 3A, 84
HK 1180 DATA 00,44,75,08,FE,06,22,44,FE,06
£ 1190 DATA 22,44,80,3E,22,44,FF,75,07,C6
HN 1200 DATA 06,22,44,07,EB,0C,80,3E,22,44
NN 1210 DATA 08,75,05,C6,06,22,44,00,47,46
FE 1220 DATA E2, AB, EB, 02, EB, 9A, 80, 3E, 22, 44
CH 123Ø DATA ØØ,75,Ø6,FF,Ø6,1E,44,EB,1E,8Ø
KP 1240 DATA 3E, 22, 44, 02, 75, 06, FF, 06, 20, 44
16 125Ø DATA EB, 11, 8Ø, 3E, 22, 44, Ø4, 75, Ø6, FF
00 1260 DATA ØE, 1E, 44, EB, Ø4, FF, ØE, 20, 44, B8
ID 1270 DATA 01,0C,8B,0E,1E,44,8B,16,20,44
LL 1280 DATA CD, 10, FE, 06, 0E, 44, BF, 0D, 00, BE
KL 1290 DATA ØE, ØØ, B9, ØE, ØØ, 8Ø, BC, ØØ, 44, Ø2
PI 1300 DATA 75,00,C6,84,00,44,00,FE,85,00
F# 1310 DATA 44,4F,4E,E2,EC,80,3E,00,44,00
OK 1320 DATA 75,02,EB,9C,1F,CB,/
```

### Program 3. The Snowflake Sweep

AH 200 IF SN(0)=0 THEN 140 KE 210 IF INKEY\$="" THEN 210

AF 23Ø DATA Ø,Ø,1,Ø,1,7,Ø,1Ø,Ø,Ø,Ø,Ø,2,1,2

PD 220 GOTO 60

For error-free program entry, be sure to use "The Automatic Proofreader," Appendix B.

DE 10 DIM DX(11), DY(11): KEY OFF MC 20 FOR N = 0 TO 6: READ SD(N), RD(N):LN(N)=1!/3! :NEXT:LN(2)=SQR(LN(1)) LE 30 A=0:FOR D=6 TO 11:DX(D)=COS(A):DY(D)=SIN(A) JN 40 A=A+.52359879#: NEXT EX 5Ø FOR D=Ø TO 5:DX(D)=-DX(D+6):DY(D)=-DY(D+6): NEXT: X1=534: Y1=147: TL=324 06 60 CLS: SCREEN Ø AB 70 PRINT"ENTER NUMBER OF CYCLES ( 1 - 4 )" OR ENTER A ZERO TO QUIT: ";N SK BØ INPUT " C JA 9Ø IF NC = Ø THEN END DA 100 IF NC > 4 THEN 60 OP 110 CLS: SCREEN 2 CL 120 X=534:Y=147:TL=324:PSET (X,Y),1 CD 130 FOR C=0 TO NC:SN(C)=0:NEXT #! 140 D=0:L=TL:NS=0:FOR C=1 TO NC: I=SN(C):L=L\*LN (I):J=SN(C-1):NS=NS+SD(J):IF NS MOD 2 = 1THEN D=D+12-RD(I):GOTO 160 SE 15Ø D=D+RD(I) ES 160 D = D MOD 12OL 170 NEXT BN 180 X=X+1.33\*L\*DX(D):Y=Y-.5\*L\*DY(D):LINE -(X,Y ),1:SN(NC)=SN(NC)+1:FOR C = NC TO 1 STEP -1: IF SN(C) <> 7 THEN 200 06 19Ø SN(C)=Ø:SN(C-1)=SN(C-1)+1:NEXT

### Simple Animation

Joseph Juhasz

If you're already familiar with IBM BASIC programming fundamentals, take a look at this article, which describes a method for animation on the hi-res graphics screen and includes a ready-to-run sample program. You'll need BASICA on the PC or Cartridge BASIC on the PCjr. The writer, founder and president of PCsoftware in San Diego, California, has designed or coauthored three commercial graphics packages for the IBM PC.

The surging popularity of microcomputers, especially the IBM PC series (PC, PC XT, and PCjr), has spurred additional interest in computer graphics and graphics software. Witness the increase not only in numbers, but also in the capabilities and ease of use, of commercially available graphics packages. There are products for drawing and animation, business charts and graphs, software that combines spreadsheets and database managers with business graphics, and even programs which coordinate animated presentations with slide shows and business charts.

One of the nicest features of the IBM PC and PCjr is the ease with which you can create color graphics. On most earlier business systems, color graphics either were not available or were extremely difficult to implement. On these earlier computers, graphics programs had to be written in machine language, a very difficult and time-consuming process.

**Advanced Graphics Commands** 

Today, systems such as the PC XT, PCjr, and Compaq have graphics capabilities as part of their normal configuration (the PC itself, however, still must be upgraded with a color/graphics adapter). Included with these systems is an advanced version of Microsoft BASIC. It's also available as an option on the PCjr. Known as BASICA on the PC or Cartridge BASIC on the PCjr, this language makes it possible for novice programmers to create graphs and drawings with such simple statements as LINE, CIRCLE, DRAW, COLOR, and PAINT. Two additional

BASICA/Cartridge BASIC statements—the graphics GET and PUT—let you manipulate graphics images on the screen.

The GET statement literally gets a graphics image off the screen and stores it in an array. Here's its format:

### GET (X1,Y1)-(X2,Y2), ARRAY(Z)

In this example, X1 and Y1 are the coordinates of the upper-left corner of the graphics image. Likewise, X2 and Y2 are the coordinates of the lower-right corner of the graphics image. This is the same coordinate system used by the LINE statement for screen modes. In fact, as you can see, the GET statement itself is very similar in format to the LINE statement.

ARRAY(Z) is simply the name of the array variable. You can, of course, use any array variable you choose.

The PUT statement, reasonably enough, is the opposite of GET. It puts the graphics image from the array back onto the screen. Here's the format of the PUT statement:

### PUT (X1,Y1),ARRAY(Z) [,option]

Here, X1 and Y1 are the coordinates of the upper-left corner of the *new location* of the graphics image—in other words, where you want to move the image. *ARRAY(Z)* again is the array variable in which the image is stored.

Preserving the Background

The last parameter, [option], can be PSET, PRESET, OR, AND, or XOR. The default is XOR (exclusive OR). Each option results in a different interaction between the image being PUT and the screen background.

PSET simply puts the image on the screen.

**PRESET** puts the reversed image on the screen like PSET—sort of like a photographic negative. An array value of 0 results in the corresponding point on the screen being displayed in color 3, and vice versa. An array value of 1 results in color 2, and vice versa.

**OR** superimposes the image onto any existing image on that part of the screen.

AND transfers the image only if another image already exists at that screen location.

**XOR**, the default, is particularly handy for animation. When you PUT an image twice somewhere on the screen, it lets you move the image without erasing the background. For

instance, assume you're writing a program to move the image of an airplane across a cloud-filled sky. It would be embarrassing if the airplane cut a swath as it flew, erasing portions of the clouds behind. With XOR, the clouds—and anything else in the background—are restored as the airplane image passes by.

**Creating Animation** 

The program listed at the end of this article, "Z-Float," demonstrates how GET and PUT are used to create simple animation on the PC or PCjr. (Z-Float is a small portion of *PCcrayon*, a graphics drawing and animation package from PCsoftware.)

Z-Float works by repeatedly erasing the image from its old location and then redrawing it at a new location. This is the basic concept for computer animation and is analogous to the sequential pictures drawn by cartoon animators. Z-Float shows a fish swimming across the screen.

When you run the program, you'll notice that the fish flickers as it moves. This is an unavoidable disadvantage of BASIC. Interpreter BASIC just isn't fast enough to erase and redraw the image without a visible delay. Z-Float can be speeded up somewhat by removing the comments and combining lines 5370–5420, but the best solution is to use the BASIC compiler. That's why almost all commercial software is either compiled or written in machine language.

**Program Description** 

When entering the program, be extra careful to type the correct values for X1, Y1, X2, Y2, X3, and Y3. Incorrect values may cause program errors, most likely because the routine will attempt to place the image off the screen.

Line(s)	Function
5040-5090	Initialize variables and prepare for graphics work.
5110-5130	Place an image on the screen to move.

5150-5190 The Z-Float routine must know what image to move, where to move it, and how quickly to move it. The image is defined by giving the coordinates of the upper-left and lower-right corners of an imaginary rectangle surrounding the image.

X1,Y1 = upper-left corner.X2,Y2 = lower-right corner.

X3,Y3 = new location of upper-left corner of the image.Variable FLOAT.SPEED can be set from 0 to 10 to vary the speed.

5220 Gets the image to be moved from the screen and stores it in the array variable PIC 5240-5320 Calculate the path of movement. First, determine the straight-line distance: DIST = SQR (DX\*DX+DY\*DY)A delay time is set according to FLOAT.SPEED. Next, determine the number of times to show the image during its movement from start to finish. Finally, calculate the distance to be moved for each step in both the x and y directions. Try different values of FLOAT.SPEED to get different effects. 5340-5420 The heart of the animation routine. It repeatedly draws and erases the image at different locations between the starting and ending points. 5380 Uses the SOUND statement to delay the movement. Removes this line for maximum speed. SOUND 32767,18\*DELAY! causes an inaudible sound of DELAY! seconds. SOUND 32767,1 pauses the program until the previous SOUND is done, thus a delay of DELAY! seconds. 5390 Erases previous image by using PUT to place an image atop itself. 5400 Determines the next stop in the animation process. To insure accurate movement, notes the variables X!, F.DX!, Y!, and FD.Y!. 5410 Places the image at its next position. 5440-5460 Place the image at its final location. **Z-Float** For error-free program entry, be sure to use "The Automatic Proofreader," Appendix B. \*\*\*\*\*\*\*\*\*\*\*\*\* KI 5000 PL 5010 Z-FLOAT KO 5Ø2Ø \*\*\*\*\*\*\*\*\*\*\*\*\* 11 5030 ' KM **5040 '\*\*---**initialization---\*\* MD 5050 DEFINT A-Z '-mak e all variables integer for speed 60 **5969** KEY OFF: FOR I=1 TO 10: KEY I."": NEXT '-turn off function keys

'-switch to Color Graphics Adpt.

'-set medium res, graphics mode

DN 5070

NF 5Ø8Ø

GOSUB 6070

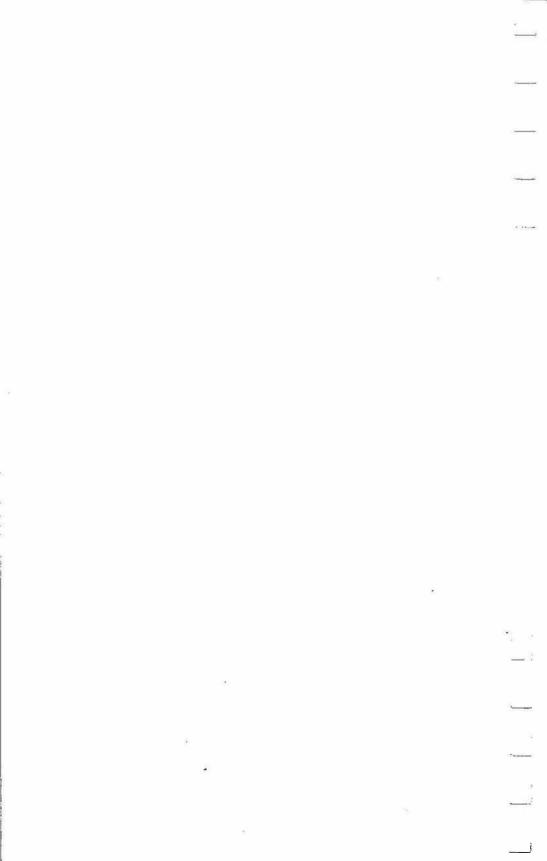
SCREEN 1, Ø

```
CI 5090
         DIM PIC (800)
        '-dimension array to hold image
HB 5100 '
NK 5110 '**---place images on screen for demo----*
DP 5120
         CLS
EF 513Ø
         X=22Ø:
                  Y=100:
                          GOSUB 6000
         '-draw fish at (220,100)
IN 514Ø '
PD 5150 '**---set variables to call float---**
         FLOAT. SPEED=9
                                         '-Set Floa
EC 5160
       t Speed from Ø(slow) to 1Ø(fast)
                                         '-Upper Le
PD 5170
         X1=215:
                   Y1=95
           Corner
NK 5180
         X2=285:
                   Y2=135
                                         '-Lower Ri
       ght Corner
                                         '-NEW LOCA
EN 5190
         X3=1Ø:
                   Y3=10
       TION - Upper Left Corner
HD 5200 '
NK 5210 '**---get the image into array PIC---**
KP 5220
         GET (X1, Y1) - (X2, Y2), PIC
IN 523Ø '
FL 5240 '**---do calculations to determine moveme
       nt path---**
DN 525Ø
         DX=X3-X1
                                         ' -DX
           = distance from X1 to X3
A8 526Ø
         DY=Y3-Y1
                                         7-DY
           = distance from Y1 to Y3
                                         '-DIST
HN 5278
         DIST=SQR (DX*DX+DY*DY)
           = distance from X1.Y1 to X3.Y3
CE 528Ø
                                        '-DELAY!
         DELAY!=(10-FLOAT.SPEED)/100
           = delay time using FLOAT.SPEED
MI 529Ø
         FLOAT.STEPS=DIST/10
                                         '-FLOAT.st
       eps = number of steps during float
HP 5300
         F.DX!=(DX/FLOAT.STEPS)
           = x distance for each step
CI 5310
         F.DY!=(DY/FLOAT.STEPS)
           = y distance for each step
IL 532Ø
         X!=X1: Y!=Y1
10 5330 "
KH 5340 '**---cause image to 'float' across scree
             ---*
       n by
OM 5350 '**---repeatedly drawing it at new locati
             ---**
       on
HM 536♥ '**---and erasing it from the old locatio
             ----
NN 537Ø
         FOR I=1 TO FLOAT. STEPS
DE 538Ø
            IF DELAY! > THEN SOUND 32767, 18 DELAY
                             '-delay
       !: SOUND 32767,1
K6 539Ø
           PUT (X!,Y!),PIC
           '-erase previous image
```

```
8F 54ØØ
            X!=X!+F.DX!: Y!=Y!+F.DY!
           '-determine new location's X.Y
           PUT (X!,Y!),PIC
06 541Ø
           '-place image at new location
CM 5420
         NEXT
IA 5430 '
IN 5440 '##---now place at final location---##
EK 545Ø
         PUT (X!,Y!),PIC
           '-erase previous image
MI 5460
          PUT (X3.Y3).PIC
           '-place image at final location
JH 547Ø '
LH 548Ø '**---END---**
JI 549Ø END
IJ 5500 *
IN 5510 '
MK 6000 '##--- Draw fish using X, Y ---##
EJ 6010
           CIRCLE (X+10,Y+10),3,1,,,1
NP 6020
           DRAW "bd515h114u1e1u1e4r1e1r1e1r1e1r2e
       1r9e4r12d2g4r2f1r2f1r2f1r3f1r3"
           DRAW "e2r1e1r1e1r4g2d1g3d1g1f1d1f3d1f2
CD 6030
       14h111h111h215d312h212g112g113g1
IH 6949
           DRAW "16g115f415h51313h114h111h1r6e1r1
       e2":
              PAINT (X+15, Y+1Ø), 1
NH 6Ø5Ø
           CIRCLE(X+12, Y+12), 8, Ø, 5.5, 1, 1:
       r10c2r16f1r8": PRESET(X+9,Y+11),2
JC 6060 RETURN
OB 6070 '**--- Switch to Color Graphics Adapter -
        --**
HI 6080
           DEF SEG=&HB8ØØ
           POKE Ø, &H33: COLGRAF= (PEEK (Ø) =&H33)
FN 6090
           POKE Ø, &H55: COLGRAF= (PEEK (Ø) =&H55) AN
CH 6100
       D COLGRAF
           IF NOT COLGRAF THEN CLS: LOCATE 12,10:
KJ 6110
        PRINT " Need Color Graphics Adapter ": E
       ND
           DEF SEG=0: I=PEEK(&H410): POKE &H410, {
16 6120
        I AND &HCF) OR &H2Ø
KJ 6130
           SCREEN Ø: SCREEN 1,0: WIDTH 40: LOCATE
         ,,Ø,6,7
10 6140
           COLOR 17,1: CLS
JB 615Ø RETURN
```

### CHAPTER FIVE

## All About Programming



# Simple Assembling with IBM DEBUG

Tim Victor

You don't need to buy an expensive assembler to write short machine language programs on an IBM PC or PCjr—a copy of PC-DOS already contains the basic tools you need. This article, which assumes some familiarity with hexadecimal numbers and machine language theory, shows how to make the most of the DEBUG utility when you're ready to tackle 8088 machine language.

Tucked away on the DOS Supplemental Programs disk that came with your copy of PC-DOS is a file called DEBUG. DEBUG is a simple but powerful development tool for exploring your computer and writing short machine language (ML) programs. It includes a miniassembler, which converts assembly language instructions into machine language directly in memory, and a disassembler, which allows you to reverse this process and examine ML programs already in memory. DEBUG also has trace and breakpoint functions for testing ML programs, utilities for loading and saving programs on disk, and several other valuable features. Using these tools, we'll show how to write a small ML program.

To get started with DEBUG, boot up DOS from your master disk. When the DOS prompt A> appears, insert the DOS Supplemental Programs disk into drive A:, type DEBUG, and press Enter. DEBUG loads and runs, replacing the DOS prompt with its own prompt, a hyphen (–). You can return to DOS at any time by putting your master disk back in the

drive, typing Q for Quit, and pressing Enter.

Since you should preserve your DOS Supplemental Programs disk as an archival backup, let's ask DEBUG to copy itself onto another disk. You could use the DOS COPY command, but using DEBUG is a good way to learn how to load and save machine language program files.

Cloning DEBUG

DEBUG has three commands for disk operations: L (Load), W (Write), and N (Name). N creates a data structure called a *file control block* (FCB) that DOS uses for all disk operations, including DEBUG's Load and Write. The FCB contains the name of a file, along with such information as size and file organization. To learn more about the FCB, consult Appendix E of the DOS 2.00 Manual, or Chapter 6 of the DOS 2.10 Technical Reference Manual.

The first step in backing up DEBUG is to load another copy of it into memory. Type *N DEBUG.COM* and press Enter. (You need to include the .COM extension because DEBUG doesn't make any assumptions about the file type.) DEBUG responds with another hyphen. Next, type *L* and press Enter. The disk drive whirs, and then another hyphen appears. You've just loaded a second copy of DEBUG.

Remove the *Supplemental Programs* disk and replace it with a formatted disk that you'll be using for ML programs. Type W and press Enter. The drive comes on again, and then DEBUG displays the message *Writing 2E80 bytes* and the hyphen prompt. You now have a copy of DEBUG.COM on your ML disk.

A Sample Program

Let's try assembling a program with DEBUG. Start by typing A 100 to start assembling at address 100H. (IBM programmers generally denote hexadecimal numbers by appending an H to the number. All input and output with DEBUG is expressed in hexadecimal.) DEBUG responds with xxxx:0100, where xxxx is a four-digit hexadecimal number. This number is the current value of the code segment register. It's of minor importance right now and will be discussed in detail later.

Now type in the following program. DEBUG displays the memory address of each instruction for you. All you need to enter are the instructions.

MOV AH,09 MOV DX,109 INT 21 INT 20 DB "HELLO THERES" Press Enter to leave the assembler. This program is the ML equivalent of everyone's first BASIC program:

#### 10 PRINT "HELLO THERE"

The ML version looks quite a bit longer, but it would be even more involved if it weren't for the INT 21H instruction, which calls a DOS function routine (Print String) by executing a software INTerrupt. Before calling this routine, the program takes two preparatory actions. The first instruction loads the AH register (an internal 8088 register) with the value 9. In 8088 machine language, instructions with two operands like MOV AH,09 operate from right to left—just as A=9 in BASIC moves the value 9 into the variable A. You specify the destination operand first, then the source operand. This might seem a little backward, but it's a common convention and you'll soon adjust to it.

AH is the high (most significant) byte of AX, the 16-bit (two-byte) accumulator register of the 8088. When a program calls Interrupt 21H, the value in AH indicates the function you're asking DOS to perform. Function 9, Print String, displays a string on the screen, starting with the character at the address contained in the DX register and ending with the character \$. The second instruction moves the address 109H into the DX register. The last instruction, INT 20H, ends the program by returning control to the program that called it—in this case, DEBUG.

Finally, a create to print is created using DB, a pseudo-opcode (*pseudo-op*). When the assembler sees a pseudo-op such as DB, it performs a function instead of generating code. This particular pseudo-op tells the assembler to store bytes of data in memory, beginning at the current location. The data can be either a list of hexadecimal numbers between 00 and FF, separated by spaces or commas, or a quoted string, as shown above. If the data is a string, the ASCII code for each character is entered in memory. The dollar sign at the end of the string is very important. Without this *delimiter*, the Print String function will keep printing whatever bytes it happens to find in memory following the message. It might be a long time before it comes across a \$ and stops.

### 8088 Memory Addressing

Now that the program is in memory, we can use the disassembler to examine it. Type *U* for Unassemble, and DEBUG displays several rows of text on the screen (the number of rows differs between 40- and 80-column displays). Notice that the disassembled code is aligned in four columns. The first column shows the address of each instruction as two four-digit hexadecimal numbers separated by a colon, just as was displayed when you entered the program. The first four-digit number is the current value of the code segment register mentioned before, and the second is the value of the *instruction pointer*. To understand why two registers are needed to point to a single memory location requires some understanding of the 8088's addressing scheme.

The 8088 microprocessor can access up to one megabyte (1024K) of memory using 20-bit addresses. However, for compatibility with older Intel processors, the 8088 has only a 16-bit instruction pointer. Because a 16-bit (four hexadecimal digit) register can only have values between 0 and 65,535, another register, the code segment register, is needed to address the entire 1,048,576 bytes allowed by the 8088. The code segment register is also a 16-bit register, but instead of addressing individual bytes, it points to blocks of 16 bytes, called *paragraphs*. Any five-digit hexadecimal address that ends in a zero is the beginning of a paragraph. For example, the byte of memory at 5D320H is at the beginning of the paragraph addressed by a segment register containing 5D32H.

The code segment register points to the first paragraph of a 64K block of memory called the *code segment* (CS). There are three other segments, the *data segment* (DS), *stack segment* (SS), and *extra segment* (ES), plus a register that points to the beginning of each. In simple programs, however, all the segment registers usually have the same value as CS. To find the next byte of code to be fetched, the value in the instruction pointer is added to the address of the beginning of the code segment. The physical address of this byte can be found with this formula:

### Physical Address = IP+(CS\*16)

The effect of organizing memory this way is that a programmer doesn't have to know where the program will be loaded. When DOS loads a .COM program, it starts the code

segment at the beginning of any available paragraph in memory. The program is loaded at an offset of 100H bytes above the start of the segment and the instruction pointer is set to 100H. The four segment registers, CS, DS, SS, and ES, all point to the start of the code segment.

The second instruction of the example program moves an address, 109H, into DX. This address is an offset into the current data segment. The string to be printed is located at an offset of 109H only if the data segment register is equal to the code segment register and the program starts at offset 100H. In practice, the CS register is rarely changed except by DOS and needs little or no attention in most programs.

**Displaying Binary Code** 

The second column of the disassembled listing on the screen contains four- or six-digit hexadecimal numbers. These are the contents of the memory locations, the binary code which the 8088 can execute. Notice that the first MOV instruction is one byte shorter than the second. The first instruction only loads half of a 16-bit register (AH is the upper half of AX), so the data occupies one byte, but the second MOV loads all of DX, which takes two bytes of data (a word).

The third column shows the *mnemonics*—symbolic names for each opcode instruction. The fourth column displays the operands. This program consists of four opcodes: two MOV instructions followed by two INT instructions. Notice that the DB pseudo-op doesn't show up in a disassembly. Instead of displaying your characters, DEBUG tries to convert the string into assembler mnemonics and therefore prints several meaningless instructions. DEBUG is frequently fooled this way because program instructions and data are both stored as binary bytes. DEBUG has no way of knowing where the program ends and the data begins.

If you type another *U*, DEBUG continues to disassemble and display the next 16 or 32 bytes in memory (depending on your screen width). Since the program is only 21 bytes long, DEBUG starts displaying part of itself, still in memory from when you copied it. Type *U* 100 to disassemble from the beginning of your program again. DEBUG's U command also accepts both starting and ending addresses if you separate them with a space.

It's a good idea to save your program on disk before running it. If the program causes something unexpected, like an infinite loop or a complete system crash, it's nice to have a copy saved. Then you can load it and search for the error without typing the program again from scratch.

As before, you need to tell DEBUG the name of your file. Type *N HELLO.COM*. There's one more thing to consider—how many bytes of memory should DEBUG write to disk? When you used the W command to copy DEBUG, it wrote the same number of bytes that it had loaded, but now you're saving a new program which has never been loaded. When DEBUG loads a file, it stores the size of the file in the CX register and the four least significant bits of the BX register. The same registers are used when DEBUG writes a file. So if your program is less than 65,536 bytes long (most are), the BX register should be set to zero.

To examine and change CX, type R CX. DEBUG prints the contents of CX (probably 2E80H, left over from copying DEBUG), then prints a colon at the beginning of the next line. You can press Enter to leave the value unchanged, or type a new value. Since the new program is 21 bytes long, type 15 (the hexadecimal equivalent of 21) and press Enter. Now type W to write the program to disk. DEBUG responds with the message Writing 0015 bytes, then returns the prompt.

Running and Debugging

Now that your program is safely on disk, run it by typing *G* and pressing Enter. The screen should display *HELLO THERE*. Then DEBUG prints *Program completed normally* followed by its usual prompt. If your program completed but didn't print correctly, disassemble starting from 100H and check that all instructions are correct. If your program locked up the computer, reboot, restart DEBUG, and thank yourself for saving the program. Reload the program with N and L, then disassemble it to see what it looks like. If you don't know what's wrong, one technique is to try setting a *breakpoint*. This halts the program at a predetermined point so that you can check the contents of the registers.

For instance, to make the program stop before the INT 20H instruction, you can set one or more breakpoints. To set a breakpoint, type *G* followed by the addresses of one or more instructions in your program. If you set more than one break-

point, separate the addresses with spaces. The program begins executing, but stops when the instruction pointer equals the address of a breakpoint. DEBUG displays the contents of all registers and flags and disassembles the instruction at the breakpoint (the instruction pointed to by the instruction pointer, the next instruction to be executed). Type *G* to restart the program at the instruction that the instruction pointer references.

If you stopped your program with a breakpoint but want to restart it from the beginning, type G=100. DEBUG sets the instruction pointer to 100H (or whatever address you specify) before starting. You can also set both the starting address and one or more breakpoints. Just include the breakpoint addresses on the same command line, separating them from the starting address and each other with spaces.

Keep this in mind: Before DEBUG executes a G command, it saves the values of all the registers, including the instruction pointer. If the program runs normally, and completes by executing INT 20H, DEBUG restores all the registers. This is great if your program runs all the way from beginning to end. You just type G and your program runs again. If, however, your program has just completed after being restarted from a breakpoint, the instruction pointer now points to the location where the breakpoint was set. Typing G starts it from the breakpoint again. To run the program from the beginning, type G = 100.

### **Learning More About DEBUG**

You've now used DEBUG to load and store program files, to assemble and disassemble a new machine language program, and to execute a program. Some other useful commands we don't have room to cover are D (Dump), which displays the contents of a block of memory as hexadecimal numbers and ASCII characters; E (Enter), to examine and change the contents of individual memory locations; and T (Trace), which executes an ML program one instruction at a time, displaying all registers and flags between instructions.

As you learn more about 8088 machine language, you'll find DEBUG a big help in testing your programs. Though you might use a separate assembler when your programs get larger, DEBUG remains useful for testing and modifying the assembled programs. If you want to know more, there is a complete description of each DEBUG command in Chapter 12

of the DOS 2.00 Manual and Chapter 8 of the DOS 2.10 Manual. Information on the DOS functions and interrupts can be found in Appendix D of the DOS 2.00 Manual and Chapter 5 of the DOS 2.10 Technical Reference Manual. To learn more about machine language programming on the IBM PC and PCjr, see COMPUTE!'s Beginner's Guide to Machine Language on the IBM PC and PCjr.

### All About Batch Files

G. Russ Davies

IBM batch programs provide a convenient way to carry out a series of DOS (Disk Operating System) commands at once. This comprehensive article covers batch programming fundamentals, then shows you how to add multiple-option menus, color, and graphic displays to batch programs.

### In the Beginning

In IBM parlance a *batch* program is simply a disk file containing a series (a batch) of DOS commands. The batch file executes these commands in sequence, just as if you manually typed them yourself. Batch files are identified with the .BAT filename extension. The most familiar example of a batch program is AUTOEXEC.BAT, used to issue startup commands to configure the system to your liking. Here's what a typical AUTOEXEC.BAT file might contain:

MODE CO80 DATE TIME CHKDSK BASICA MENU

The first four commands in this batch file are familiar DOS commands to set the display mode to 80 columns (MODE CO80), let you input the date and time (DATE and TIME), and analyze the disk directory (CHKDSK). (Note that if the AUTOEXEC.BAT file doesn't include DATE and TIME, the system doesn't ask for date and time inputs when it boots.) The last command activates BASICA, then loads and runs a BASIC program named MENU. A file named AUTOEXEC.BAT differs from other batch files only in that it runs automatically when you turn on the system.

To run a batch program that doesn't run automatically, simply enter the filename at the DOS prompt (you can leave off the .BAT extension). This tells DOS to load the batch file from disk and carry out each of its commands in order. For instance, to run a program named SETUP.BAT, you would type SETUP after the DOS prompt and press Enter.

This article presents several example batch programs. (Since these are *not* BASIC programs, don't try to enter them with the "Automatic Proofreader" in Appendix B.) The DOS manual explains how to type in short batch programs using the COPY CON: command from DOS. However, for any batch program longer than a few lines, it's easier to use a word processor or any text editor that creates standard ASCII files. Most commercial programs are suitable. You can also use the EDLIN program (on the *DOS Supplemental Programs* disk), though it lacks the convenient editing features of word processors.

### **Chains and Parameters**

In the AUTOEXEC.BAT example above, the batch program ends by loading BASIC and running a BASIC program. A batch program can also end by returning control to DOS, or by running a second batch program (permitting you to *chain* two or more programs together). For instance, ending a batch program with SECOND causes the system to load and run the batch program named SECOND.BAT. You can also use COMMAND /C to run one batch program from within another. For instance, COMMAND /C SECOND runs SECOND.BAT.

Passing parameters (information) to a batch program is straightforward. Simply include the needed information after the filename when running the program. For example, typing FIRST JULIA 123 runs the FIRST.BAT program and passes two parameters to it: a string (JULIA) and a number (123). In much the same way, one batch program can pass parameters to another. Let's use an example to demonstrate parameter passing in chained programs. Enter the following batch program and save it to disk with the filename FIRST.BAT.

```
echo off
echo first_bat uses first parameter: %1
echo passes %2 and %3 to second.bat
second %2 %3
```

Now, enter the following program and save it with the filename SECOND.BAT:

```
echo second.bat uses second parameter: %1 echo passes %2 to third.bat third %2
```

Finally, enter the following program and save it with the filename THIRD.BAT:

### echo third bat uses third parameter: %1

At this point you have three batch programs, all of which expect parameters. To run the programs, enter *FIRST* followed by any three strings or numbers. Be sure to separate each parameter with a space. For instance, you might enter *FIRST PARAM/ONE &H464 IBMBIO.COM*. The FIRST.BAT program takes in all three parameters, processing the first (displaying it in an ECHO statement) and passing the other two when it runs SECOND. SECOND.BAT processes the second parameter and passes the third to THIRD.BAT.

As shown in these examples, batch programs use dummy parameters (% followed by a digit from 0 through 9) to mark the spot where the real parameter is expected. When you run a batch program, each dummy parameter is replaced by actual data in the order it's received. Thus, the FIRST.BAT program uses %1 to signify the first parameter, %2 to represent the second, and so on. Dummy parameter %0 can be replaced only by a drive designator (A or B) and filename: Don't use it unless you want to pass such information.

Be sure to keep the dummy parameter numbers straight when chaining batch programs. The dummy number represents the order *in which that program receives the data*. In the example above, FIRST.BAT received three parameters, which it represents with the three dummies %1, %2, and %3. SECOND.BAT receives two parameters, using %1 to signify the first parameter it receives, and %2 to represent the second. Likewise, THIRD.BAT uses %1 to represent its single parameter. (Note that THIRD.BAT can't use %3 for the dummy. Though you, the programmer, may think of this parameter as the *third*, it's the first one that THIRD.BAT receives.)

### **Batch Commands**

In addition to ordinary DOS commands, a batch program may include the following special batch commands: ECHO, FOR, GOTO, IF, SHIFT, PAUSE, and REM.

ECHO ON causes DOS commands to be displayed as they're performed in a batch program, while ECHO OFF turns off the display. As you saw above, ECHO can also display messages. GOTO is discussed later in this article. REM lets you include remarks, and SHIFT is used when more than ten

parameters are passed at one time.

The remaining commands (FOR, IF, and PAUSE) permit loops, conditional tests, and limited user input. The short file-copying program listed below demonstrates all three of these commands. Enter the program as listed, saving it with the file-name COPYUNQ.BAT (or any other name ending in .BAT).

REM------

REM name: COPYUNG.BAT
REM syntax: COPYUNG source-drive-letter
target-drive-letter (no colons)
REM purpose: Only unique files are

copied from source to target disk

-----

%1:

FOR %%f in (\*.\*) DO IF exist %2:%%f ECHO
%%f WILL NOT BE COPIED

PAUSE READY TO BEGIN COPIES,
FOR %%f in (\*.\*) DO IF not exist %2:%%f
COPY %1:%%f %2: /V
%2:

The COPYUNQ.BAT program automatically copies files from a source disk to a target disk, copying only those files that don't already exist on the target disk. This insures that existing files are not replaced, an improvement over DOS's COPY command, which would write over any like-named files on the target disk. To run this program, enter its name followed by the letter of the source drive and the letter of the target drive. Colons are not required after the drive letters. For instance, you would enter COPYUNQ.BAT A B when drive A holds the source disk and drive B holds the target disk. The program displays the names of files that are not copied.

### FOR and IF

COPYUNQ.BAT offers a good demonstration of FOR and IF, which work very differently from their BASIC equivalents.

Since a FOR statement can't contain another FOR statement, you can't use nested FOR loops (one FOR loop enclosed by another). FOR statements take the following general form:

### FOR %%variable IN (set) DO DOS command

The set value after IN represents a group of files and must be some variation of a filename and extension. This parameter determines which disk files the FOR loop will affect. Since the pattern-matching symbols \* and ? can be used, you may define this group to be very broad or very selective. The program shown above uses the statement *IN* (\*.\*) to affect the broadest possible group—every file on the disk. In other cases, you might use IN (\*.BAS) to affect all files ending with .BAS, IN (ABC\*.\*) to affect all files starting with the characters *ABC*, and so on.

The first FOR statement in COPYUNQ.BAT (FOR %%f IN (\*.\*) DO) affects every file on the disk. As the FOR loop executes, the variable %%f represents each filename in order. Translated into plain English, this statement means cycle through every filename on the source disk, using %%f to represent each filename in turn.

IF can perform only a few tests. One of these (IF EXIST filename) tests whether a given file exists on the disk. Now you can understand the second part of the FOR statement (IF EXIST %2:%%f). The %2 parameter is a dummy, replaced by the second drive letter you entered when running the program. And the variable %%f is replaced by actual filenames when the program runs. In plain English, this statement means if the current filename exists on the disk in the target drive....

Batch programs don't have the equivalent of BASIC's THEN statement (THEN is implied). But in other respects IF processing works much as it does in BASIC. Statements that come after the IF test (on the same line) are performed when the IF test is true and are skipped when the test is false. Consequently, in COPYUNQ.BAT, the ECHO command (which prints [filename] WILL NOT BE COPIED) executes only when the file in question exists on both the source and target disks.

Once you understand that much of COPYUNQ.BAT, the rest is not hard to decipher. PAUSE makes the system stop and display the message *Strike any key when ready*. This is the only batch command that allows user input. Unfortunately,

your choices are severely limited: You can continue only by pressing a key (perhaps after changing disks) or end the program by pressing Ctrl-Break. In later parts of this article, you'll see how to expand this number of options.

### NOT and ERRORLEVEL

The second FOR line in COPYUNQ.BAT has a FOR loop and an IF test very similar to the first. However, in this case NOT reverses the logic of the IF test. When the named file *does not exist* on the target disk, the IF test is true and the file is copied.

In addition to testing EXIST (with or without NOT), IF can test two conditions: the equality symbol (==) and ERRORLEVEL. The equality symbol tests whether two strings are identical. ERRORLEVEL is always a number, ordinarily used to pass information from one program to another (indicating whether the first worked successfully and thus set ERRORLEVEL to the expected value). ERRORLEVEL is discussed in more detail a bit later.

As shown in these brief examples, batch programs can be very powerful: IF lets you pick only the files you want, and FOR lets you repeat commands until the whole task is done. In one sense, the lack of opportunity for user input is an advantage: The entire procedure is automated, and you don't need to understand anything except how to type in the program name. On the other hand, batch programming can seem rigid, limiting, and visually quite dull. This next section improves on that situation, offering program examples and a routine that adds colorful graphic displays and multiple-option menu selection to batch programs.

**Onward and Upward** 

As you've seen, IBM batch programs can be extremely powerful. The batch commands FOR, IF, and GOTO permit program loops, conditional tests, and program branching. You can also chain two or more batch programs together and pass information from one to another.

But batch programs have limitations, too. Visual displays are often unexciting, consisting of single-color alphanumerics (no graphics characters, for instance), and user input is even more restricted. The PAUSE command offers only two op-

tions: continuing after the pause or ending the program. This virtually rules out complex, interactive programs that let you select from several different options to perform various tasks.

### **Adding Choices**

The "CHOOSE.COM" program below provides the equivalent of a new batch command. As the name suggests, CHOOSE lets you make a choice. It can be used by itself to request a yes/no response, or with additional information to offer several different options. Since CHOOSE.COM is a machine language program, there's a BASIC filemaker program included which creates it for you. Type in and save Program 1 as listed at the end of the article, then run it. Once that's done, you can try out the simpler yes/no form of CHOOSE.

You'll remember that any batch program named AUTO-EXEC.BAT loads and runs automatically when you boot the system. An AUTOEXEC.BAT program which doesn't include the DOS commands DATE and TIME won't prompt you to enter the date and time (as normally happens when you boot up). Though it's often valuable to have correct date and time information on new files, there are also many times when you don't need it.

The following short batch program lets you choose whether to add date and time settings. Enter it as listed, using the EDLIN program (on the DOS Supplemental Programs disk) or any word processor or text editor that produces standard ASCII output. Keep in mind that since this and other examples are not BASIC programs, you don't need to enter them with the IBM Automatic Proofreader from Appendix B.

Once you have entered this program, save it with the filename AUTOEXEC.BAT. Because the program calls CHOOSE.COM, you *must* save it on a disk that contains CHOOSE.COM.

```
echo off
MODE CO80
echo Do you wish to set the date/time?
rem press Y,y,N, or n to answer
CHOOSE
IF ERRORLEVEL 1 GOTO :setdt
goto :next
:setdt
```

date
time
:next
CHKDSK
BASICA MENU

After saving this program, run it by rebooting the system (press Ctrl-Alt-Del or enter AUTOEXEC). When used without parameters, CHOOSE checks for a yes/no response, permitting uppercase as well as lowercase Y and N (it's not necessary to press the Enter key after typing Y or N). Other responses (except Ctrl-Break) cause the prompt message to be displayed until a valid choice is made.

### ERRORLEVEL Is a Variable

After you respond with yes or no, CHOOSE passes this information to the batch program via ERRORLEVEL. As explained earlier, ERRORLEVEL is a special variable you can test with IF. In this example, CHOOSE sets ERRORLEVEL to 1 when the response is yes, and 0 when the response is no. The GOTO command then branches appropriately. Note that GOTO branches to a *destination label*, which is a colon followed by a string. This program uses the labels :setdt and :next. Don't confuse the label :next with BASIC's NEXT statement (which doesn't exist in batch programming).

In this case, ERRORLEVEL can have only one of two possible values, but it can take higher values as well (see below). When testing ERRORLEVEL with IF, keep in mind that the IF ERRORLEVEL statement is true when ERRORLEVEL is greater or equal to the number being tested. If you tested for 0 first in this program, ERRORLEVEL would always be 0 (1 and 0 are both greater than or equal to 0). When testing ERRORLEVEL, you must always test for higher values before testing for lower ones.

**Multiple Options** 

Most utility programs offer a variety of options. Typically, they display a menu with a list of options, and you choose the option you want by pressing a certain key. CHOOSE makes it easy to present such menus within a batch program. First, display the options on the screen, then use CHOOSE followed by a list of the keys you wish to test. For instance, the statement

CHOOSE ABC checks the A, B, and C keys and returns appropriate values in ERRORLEVEL. The ERRORLEVEL value corresponds to the position of the key in the list after the CHOOSE command. Thus, after the program performs CHOOSE ABC, ERRORLEVEL equals 1 if A was pressed, 2 if B was pressed, and so on.

When using CHOOSE with several option keys, it's critical to list the keys in the right order. Since you must always test for higher ERRORLEVEL values before testing for lower ones, you'll want to put the most likely (or most speed-critical) options at the *end* of the option key list. This assigns higher ERRORLEVEL values to the more important options.

### **Entering FILES.BAT**

Program 2, "FILES.BAT," which is listed at the end of the article, demonstrates multiple-option selection as well as a colorful, attractively formatted menu and help panel. It sorts any disk directory by file size, date, filename extension, or alphabetical order, and can also create separate batch files for mass DOS operations. Entering the program requires several steps:

- Make sure your disk contains the system file called ANSI.SYS. If necessary, copy ANSI.SYS from the DOS disk with the COPY command. This file contains the screen/ keyboard driver used for graphics displays and temporary key assignments.
- 2. Make sure your disk contains a file named CONFIG.SYS that includes the statement DEVICE=ANSI.SYS. If your disk already has a CONFIG.SYS file, add that statement to the file with EDLIN or another text editor. If your disk doesn't have a CONFIG.SYS file, create one by entering these lines:

### COPY CON:CONFIG.SYS DEVICE=ANSI.SYS

Press the F6 key to end the file, then press Enter. Your disk now contains the necessary CONFIG.SYS file.

3. Using EDLIN or some other text editor, enter Program 2 as listed below and save it on disk with the name FILES.BAT. (Since this is *not* a BASIC program, don't try to enter it with the IBM Automatic Proofreader.) Several lines in the listing contain the characters {CTRL-V}. The braces indicate that this is a special control character which you must

enter by pressing a combination of keys. Do *not* type the braces. Instead, wherever you see {CTRL-V} in the listing, hold down the Ctrl key and press the V key. On the screen, you'll see the wedge-shaped control character that precedes special ANSI.SYS screen or keyboard instructions. Type everything else in Program 2 exactly as it appears. (Note that in many instances, two bracket characters will appear side by side, as in [[. This is *not* a misprint—enter both brackets.)

4. In the same manner, type in Program 3 as listed and save it on disk with the name FILES.MNU (use this filename only). This file is graphics data for the menu. Whenever you see {CTRL-V} in the listing, enter Ctrl-V as described above. A number enclosed in braces indicates a graphics character (the number is an ASCII code) which you must enter with the Alt-keypad technique on the PC and by another method on the PCjr. For instance, where the listing contains {218}, hold down the Alt key, then type the characters 2, 1, and 8 on the numeric keypad. When you release the Alt key, character 218 appears on the screen. On the PCjr, hold down Alt, press Fn-N, then enter the numbers as on the PC. After all three numbers are entered, release the Alt key; the character will appear on the screen.

When the braces enclose two numbers separated by a space, several characters are needed—the first value shows how many characters to enter, the second is the ASCII code. For example, where you see {5 196}, use the above procedure to enter character 196 five times. Where you see the letters *SP* followed by a number and enclosed in braces, you should type the space bar the indicated number of times. For instance, {SP 16} means to type 16 spaces.

- 5. Enter Program 4 as listed, using the technique described above, and save it on disk with the filename FILES.HLP (make sure to use this filename). This file contains graphics data for the help screen.
- 6. Enter a batch program that contains nothing but a REM statement and save it on disk with the filename QUIT.BAT. This can be done with a text editor or by entering these statements from DOS:

COPY CON: QUIT.BAT REM ANYTHING

Now press the F6 key, followed by Enter.

7. Activate BASIC and type in Program 5. Since this program is listed in BASIC, enter and save it using the IBM Automatic Proofreader from Appendix B. You must save this program with the filename FILEGRP.BAS.

8. Finally, before using FILES.BAT, check your disk to make sure all the necessary files are present. It *must* contain CHOOSE.COM, ANSI.SYS, CONFIG.SYS, FILES.BAT, FILES.HLP, FILES.MNU, FILEGRP.BAS, and QUIT.BAT. The program will not work correctly unless all these files are on one disk and named as shown here. Note that the FILEGRP option (see below) also requires BASIC.

### Using FILES.BAT

Before you run this program, reboot the system by turning the computer off and on or by pressing Ctrl-Alt-Del. This guarantees that the ANSI.SYS driver is present. To run FILES.BAT, type *FILES* after the DOS prompt and press Enter. Most of the program is self-explanatory—after all, that's what menus and help screens are for—so we won't describe every option.

The FILEGRP option lets you create a separate batch file (named FILEGRP.BAT) for performing operations on a group of files. Every line in FILEGRP.BAT consists of a filename from the subject disk and four dummy parameters in this order:

### %1 filename.extension %2 %3 %4

Dummy parameters are replaced by actual parameters you supply when running FILEGRP.BAT. This makes it easy to perform the same operation (copy, print, delete, and so on) on a large group of files. After using the FILEGRP option, exit to DOS and use a word processor or text editor to edit FILEGRP.BAT as needed, deleting the names of any files you don't want to include in the operation. Then run FILEGRP.BAT by entering its name followed by the desired parameters. The first parameter can be any DOS command; the rest will be parameters relevant to that command. For instance, you might enter FILEGRP COPY B: /V to copy the files listed in FILEGRP.BAT onto drive B. Incidentally, BASIC doesn't provide any way to set ERRORLEVEL.

**Advanced Batch Programming** 

FILES.BAT employs several techniques you may find useful. The DOS command BREAK ON makes the system respond to Ctrl-Break in more instances than normal. The TYPE command is used to display graphics like the menu and help screen. TYPE creates such displays much faster than the DOS ECHO command (you could also use COPY).

The ANSI.SYS driver assigns the lowercase keys *a*, *s*, *d*, *e*, *b*, and *i* to their uppercase equivalents to reduce the amount of testing required. The F1 and F10 keys are assigned to keys H and X, respectively, so those function keys perform their usual HELP and EXIT roles. After CHOOSE accepts a response, the modified keys are restored to their original definitions. Pressing Ctrl-Break while CHOOSE is active (or pressing Y in response to *Terminate batch file?*) leaves these keys reassigned. To avoid this effect, you should normally exit by pressing F10.

The F10 (EXIT) function uses a trick to perform a quick exit. It simply runs QUIT.BAT, a batch program that consists of a do-nothing REM statement. When any batch program ends, it ends all preceding batch programs as well. Note that since ECHO OFF is in effect when QUIT.BAT is called, the REM is

not displayed.

Batch commands are not particularly fast. To optimize speed, structure the program so that the most-often used (or speed-critical) routines are closest to the place you're branching from. The fewer program lines that a GOTO has to skip over, the quicker it executes. You can also speed up batch programs by using extra disk buffers as explained in the DOS Manual. REM statements slow batch programs drastically; if you want to document the program, store your comments in a separate file.

In some cases it's useful to test for the absence of a parameter. For instance, you might want to reprompt the user with a message like *You must enter more information*. This can be done with a statement such as IF .-%1. GOTO .NOPARM. This line means if a dot equals the parameter plus a dot, then go to the no-parameter routine. The IF test is true only when no parameters have been entered.

### Program 1. CHOOSE.COM Filemaker

For error-free program entry, be sure to use "The Automatic Proofreader," Appendix B.

```
N 100 OPEN "CHOOSE.COM" FOR OUTPUT AS #1
LA 110 READ X4: IF X4="/*" GOTO 130
CA 120 PRINT #1, CHR$ (VAL ("&H"+X$));:60T0 110
18 130 CLOSE #1:END
KB 140
      DATA AØ,8Ø,Ø,3C,Ø,75,2D,9Ø,BA,6Ø,1,B4,9,C
      D, 21, B4
PC 15Ø
      DATA C, BØ, 7, CD, 21, 3C, 59, 74, F, 3C, 4E, 74, 10,
      3C, 79, 74
FG 160
       DATA 7,3C,6E,74,8,EB,E1,9Ø,BØ,1,EB,3,9Ø,B
       DATA 4C, CD, 21, 90, BA, BØ, 1, B4, 9, CD, 21, 84, C,
EL 179
      BØ, 8, CD
       DATA 21,88,C4,90,BD,0,0,45,8A,86,80,0,3C,
SN 180
      D, 74, E4
       DATA 38, EØ, 75, F3, 89, E8, 9Ø, 48, B4, 4C, CD, 21,
KC 190
      90,90,90,90
       DATA 43,68,6F,6F,73,65,20,59,20,28,79,65,
NJ 200
      73,29,20,6F
FM 210
       DATA 72,20,4E,20,28,6E,6F,29,20,2E,2E,2E,
      D, A, 24, 20
       DATA 43,68,6F,6F,73,65,20,64,65,73,69,72,
₽J 22Ø
      65,64,20,6F
       DATA 70,74,69,6F,5E,20,2E,2E,2E,D,A,24,0,
OK 230
      0,0,0
KM 240 DATA /*
```

### Program 2. FILES.BAT

```
echo off

rem Name: FILES.BAT [filename_ext]

See help panel for usage

break on

dir %1 > temp_dir

:menu

cls

type files.mnu

echo{CTRL-V}[["a";"A"p{CTRL-V}[{"s";"S"p
 {CTRL-V}[["d";"D"p{CTRL-V}[["e";"E"pf
 CTRL-V][["b";"B"p
 {CTRL-V}[["i";"l"p

echo{CTRL-V}[["i";"l"p

echo{CTRL-V}[[0;59;"H"p{CTRL-V}[[0;68;"X
 "p {CTRL-V}[[2A

choose EIBSDHAX
```

```
echo{CTRL-V}[["a"; "a"p{CTRL-V}[["s"; "s"p
   {CTRL-V}[["d";"d"p{CTRL-V}[["e";"e"p{
   CTRL-V3[["b":"b"p{CTRL-V3[["i":"i"p
echo{CTRL-V}[[0;59;0;59p{CTRL-V}[[0;68;0
   :68p {CTRL-V}[[0m
if errorlevel 8 QUIT
if errorlevel 7 goto :a
if errorlevel 6 goto :h
if errorlevel 5 goto :d
if errorlevel 4 goto :s
if errorlevel 3 goto :b
if errorlevel 2 goto :i
                 goto :e
: a
CIS
sort /+1 <temp_dir >con
pause
goto :menu
copy files.hlp con
pause
goto :menu
: d
CIS
sort /+24 <temp.dir >con
pause
goto :menu
: 5
sort /+14 /R <temp_dir >con
pause
goto :menu
: b
basic filegrp
echo ----- FILEGRP.BAT Created
   -----
pause
goto :menu
: j
CIS
dir %1 /p
pause
```

goto :menu

e cls

```
sort /+10 <temp_dir >con
Dause
goto :menu
Program 3. FILES.MNU
{CTRL-V}[[2] {CTRL-V}[[32m]
{SP 16}{218}{5 196} {CTRL-V}[{33m}
   DIRECTORY DISPLAYS MENU
   {CTRL-V}[[32m{5 196}{191}]
(SP 16) (179) (SP 35) (179)
{SP 16} {179} {CTRL-V} [[35m A
   {CTRL-V}[[32m- Alphabetical order by
   filename {179}
{SP 16} {179} {SP 35} {179}
{SP 16}{179}{CTRL-V}[[35m E
   {CTRL-V}[[32m- Ext name order{SP
   173 [ 179 ]
{SP 16} {179} {SP 35} {179}
(SP 16) (179) (CTRL-V) [[35m D
   {CTRL-V}[[32m- Date order, Yr not
   significant {179}
(SP 16) [179] (SP 35) [179]
{SP 16}{179}{CTRL-V}[[35m S
   {CTRL-V}[[32m-Size order{SP 21}{179}
(SP 16) (179) (SP 35) (179)
{SP 16}{179}{CTRL-V}[[35m B
   {CTRL-V}[[32m- Bat file creation:
   FILEGRP.bat [179]
{SP 16}{179}{SP 35}{179}
(SP 16) (179) (CTRL-V) [[35m |
   {CTRL-V}[[32m- Intrinsic order of dir
   entries [179]
(SP 163 (1793 (SP 353 (179)
{SP 16}{179}{CTRL-V}[[35mF1
   {CTRL-V}[[32m- HELP{SP 27}{179}
[SP 16][179][SP 35][179]
ESP
   16}{179}{CTRL-V}[[35mF10{CTRL-V}{[32m
   - EXIT(SP 273(179)
```

{SP 16}{179}{SP 35}{179}
{SP 16}{192}{36 196}{217}
{CTRL-V}[[31m]

### Program 4. FILES.HLP

- {CTRL-V}[[44:33m{CTRL-V}[[2J{CTRL-V}[[1m] {SP 73(201)(15 205) {CTRL-V}[[35m] DIRECTORY DISPLAY HELP {CTRL-V11[33mf16 205] [187] [SP 7] [186] [SP 2] PURPOSE: Produces a directory listing (SP 173 (186) (SP 73 (1863 (SP 12) sorted in the desired order [SP 16] [186] ESP 73 (1863 (SP 2) SYNTAX: (SP 2) FILES [d:][filename][.ext][SP 20][186] {SP 7}{186}{SP 9}(if parameters are omitted, \*.\* used) {SP 10} {186} (SP 73 [186] (SP 56] [186] {SP 7}{186}{SP 2}MENU OPTIONS:{SP 413 [ 1863 {SP 7}{186}{SP 4}A: Directory sorted ascending by filename (SP 11) [186] (SP 7) (186) (SP 4) E: Directory sorted ascending by file extension (SP 53 [ 186 ]
- {SP 73{1863{SP 4}D: Directory sorted ascending by file date (mm-dd){SP 2}{186}
- {SP 73{186}{SP 7}giving calendar order, year least significant(SP 4){186}
- {SP 73{186}{SP 4}S: Directory sorted DESCENDING by file size{SP 9}{186}
- {SP 7}{186}{SP 7}allowing quick
   determination of largest files{SP
   4}{186}
- {SP 7}{186}{SP 4}B: FILEGRP.BAT created
  as : %1 filename.ext %2 %3 %4{186}
- {SP 7}{186}{SP 7}for editing and mass file copy, erase, type, etc.{186}

- {SP 73{186}{SP 4}{I: Directory in the order of the directory entries{SP 2}{186}
- (SP 73 (1863 (SP 563 (1863
- {SP 7}{186}{SP 4}H or F1: Displays this
  help panel{SP 19}{186}
- ESP 73[186]{SP 4}X or F10: Fast exit to DOS(SP 26][186]
- {SP 7}{200}{56 205}{188}{CTRL-V}[[0m pg

### Program 5. FILEGRP.BAS

For error-free program entry, be sure to use "The Automatic Proofreader," Appendix B.

- N 10 'This program creates a batch file named FI LEGRP.BAT, using the
- LO 20 'TEMP.DIR file created by FILES.BAT. FILEGR
  P.BAT is useful for
- # 30 'group file operations such as copying, del eting, printing, etc.
- EK 40 'Each line in FILEGRP.BAT has the format: % 1 filename.ext %2 %3 %4
- % 5Ø 'Use a word processor or text editor to del ete non-participating
- ID 60 'files from FILEGRP.BAT.
- ff 70 OPEN "temp.dir" FOR INPUT AS #1'input file
- #N 80 OPEN "filegrp.bat" FOR OUTPUT AS #2'output file
- EC 90 FOR X= 1 TO 4: IF EOF(1) THEN SYSTEM'skip 4line header
- KD 100 LINE INPUT#1, X\$: NEXT
- 酬 110 IF EOF(1) THEN SYSTEM'check for end-of-file
- LA 120 LINE INPUT #1, X\$'get input line
- N 130 IF LEFT\$(X\$,1)=" " GOTO 110'skip lines beg inning with space
- & 140 Z=INSTR(X\*," "):Z=Z-1'find length of filen
- E6 15Ø PRINT #2,"%1. ";MID\$(X\$,1,Z);".";MID\$(X\$,1Ø ,3);" %2 %3 %4"'form output
- #L 160 GOTO 110'continue till end-of-file

## **Lightning Sort**

Russ Gaspard PC/PCir Version by Tim Victor

Three years ago, COMPUTE! magazine published "Ultrasort," a program for the Commodore computers. We called it the fastest sorting program ever published for any home computer. It would sort a 1000-element array in less than eight seconds.

It's been improved. "Lightning Sort" does the same thing in a breathtaking 2.1 seconds. Add this extraordinarily powerful subroutine, adapted from the Commodore version, to any of your BASIC programs where you need to alphabetize something.

For the PC and PCir.

The "Ultrasort" routine for Commodore computers (COM-PUTE!, September 1983, p. 194) isn't as fast as it could be. After disassembling the code to study the algorithm, I found several opportunities to compact the code (mainly to reduce disk loading time) and to speed up the execution time. Using the "Sort Test" program from the original article as a benchmark, my "Lightning Sort" routine sorts a 1000-element array in an average of 1.91 seconds, versus 7.8 seconds for Ultrasort. That few seconds savings isn't much. But when I tried it on random 4000-element arrays, the routine took an average of 9.9 seconds, versus 40 seconds for Ultrasort. A 400 percent speedup in execution time can be significant in applications where the sort routine is called repeatedly or in sorting very large arrays.

The time for this type of algorithm to sort an N-element array is T\*N\*Log2N on the average. Actual running time depends on the starting order of the array. Interestingly, whereas many sort algorithms run fastest when the original array is already in order, Hoare's "Quicksort" runs fastest on randomly ordered data. If you try it on an array which is already in correct order, you'll find that it takes much longer (proportional

to  $N^2$ ).

Besides speeding up the execution, the amount of RAM needed was reduced from 908 bytes to 512 bytes. By storing the variables in RAM space above the actual sorting routine rather than within the routine, the actual program storage needed on disk is only 220 bytes.

Program 1 is a BASIC program which loads the machine language Lightning Sort routine for the IBM PC and PCjr. The routine is loaded into RAM from \$FF00 to \$FFDC (decimal 65280 to 65500).

### **Sort Now**

The BASIC loader program calculates a checksum from the DATA statements to help identify typing errors, then creates a disk file named "LSORT.BAS", containing the ML routine in binary form. The demonstration (Program 2) loads this file into memory using BLOAD and sets LSORT to the address of the sort routine. This variable is needed because IBM BASIC's CALL statement will accept only a variable name for the address of an ML routine.

To see Lightning Sort in action, make sure you've created a copy of LSORT.BAS by running Program 1, then type in Program 2. Save it to the same disk as LSORT, then run it.

Program 2 BLOADs LSORT, generates a 1000-element array, sorts it, and shows how long it took.

### Details

Lightning Sort uses the first parameter in the CALL statement to find the array it will sort (refer to line 180 in Program 2). This is actually the address of the first string in the array, AA\$(1) in the demonstration program, not the address of the array itself. The second parameter, N%, tells Lightning Sort how many strings are in the array. Variable names also have to be used for parameters, which is why N% is used instead of just plain 1000. The program expects the length parameter to be an integer variable (a variable whose name ends with a percent sign).

Lightning Sort is loaded at address \$FF00 in BASIC's default segment. During a sort, the 256 bytes starting at \$FE00 are also used. To protect this memory, both programs start with the instruction CLEAR,&HFE00, which sets the top of BASIC's workspace to \$FE00.

### **Program 1. Lightning Sort Loader**

For error-free program entry, be sure to use "The Automatic Proofreader," Appendix B.

```
60 100 CLEAR, &HFE00
AC 110 ON ERROR GOTO 10000
DL 120 DEF SE6
KL 130 CHECKSUM = 0
CD 140 ADDRESS = &HFF00
BE 150 READ MLDATA
FC 16Ø WHILE MLDATA <> -1
#P 170 POKE ADDRESS, MLDATA
NE 180 CHECKSUM = CHECKSUM + MLDATA
NI 190 ADDRESS = ADDRESS + 1
BL 200 READ MLDATA
DH 21Ø WEND
II 220 IF CHECKSUM <> 22937 THEN ERROR 200
KH 23Ø BSAVE "1sort", &HFFØØ, &HDD
LD 240 END
HL 1000 DATA 85,137,229,139,118,6,139,4
IN 1010 DATA 72,185,3,0,247,225,139,86
80 1020 DATA 8,1,208,189,252,254,137,86
00 1030 DATA 2,137,70,0,252,41,192,80
ND 1040 DATA 139,94,0,139,86,2,57,211
NP 1050 DATA 127,3,233,129,0,135,211,232
AH 1060 DATA 139,0,118,5,131,195,3,235
LK 1070 DATA 246,135,211,57,211,126,31,131
CM 1080 DATA 235, 3, 232, 120, 0, 114, 244, 138
BJ 1090 DATA 15,139,71,1,135,211,134,15
JE 1100 DATA 135,71,1,135,211,136,15,137
HB 1110 DATA 71, 1, 135, 211, 235, 214, 139, 118
PF 1120 DATA 0,138,4,134,7,136,4,139
FL 1130 DATA 68,1,135,71,1,137,68,1
CM 1140 DATA 139,86,0,3,86,2,209,234
ED 1150 DATA 57,218,114,23,139,70,2,131
GL 1160 DATA 195,3,137,94,2,131,237,4
DH 1170 DATA 131,235,6,137,94,0,137,70
BK 1180 DATA 2,235,21,139,70,0,131,235
IN 1190 DATA 3,137,94,0,131,237,4,131
JE 1200 DATA 195,6,137,94,2,137,70,0
NK 1210 DATA 88,64,80,233,114,255,88,72
HG 1220 DATA 124,7,80,131,197,4,233,103
GA 123Ø DATA 255,93,202,4,0,139,118,0
BN 1240 DATA 181,0,138,12,139,116,1,58
OB 1250 DATA 15,118,2,138,15,139,127,1
DE 1260 DATA 243, 166, 116, 1, 195, 139, 126, 0
SN 127Ø DATA 138,13,58,15,195,-1
NP 10000 IF ERR <> 200 THEN ON ERROR GOTO 0
AB 10010 PRINT "Error in ML data: check for typo'
        5"
DP 10020 RESUME 240
```

### **Program 2. Sorting Demonstration**

For error-free program entry, be sure to use "The Automatic Proofreader," Appendix B.

- BF 10 CLEAR, &HFE00 : DEF SEG : CLS
- JB 20 BLOAD "Isort", &HFF00:LSORT=&HFF00
- NJ 30 N%=1000
- FH 40 DIM AA\$ (N%)
- 08 50 LOCATE 2,16 : PRINT "Creating ";N%;"random strings"
- NI 60 DEF SEG=&H40: RANDOMIZE PEEK (&H6C)
- CN 70 FOR I=1 TO N%:LOCATE 3,16:PRINT I
- 18 8Ø J%=RND(1) \*1Ø+1
- CC 9Ø A\$="":FOR K=1 TO J%
- BM 100 A\$=A\$+CHR\$(INT(RND(1) \*26+65))
- ON 110 NEXT K
- FG 120 AA\$(I)=A\$
- NN 130 NEXT I
- AM 140 CLS:LOCATE 2,16:PRINT "Any key to start so rt:"
- EG 150 AS="":WHILE AS="":AS=INKEYS:WEND
- OC 160 LOCATE 3,16:PRINT "sorting- ";
- NJ 170 SS=PEEK(&H6C)+256\*PEEK(&H6D)
- EE 180 DEF SEG: CALL LSORT (AA\$(1), N%)
- FJ 190 DEF SEG=&H40:FS=PEEK(&H6C)+256\*PEEK(&H6D)
- KJ 200 PRINT "done"
- CL 210 LOCATE 5,16:PRINT "Any key to print sorted strings"
- FO 215 AS="":WHILE AS="":AS=INKEYS:WEND
- CH 220 FOR I=1 TO N%:PRINT AA\$(I):NEXT
- HH 23Ø PRINT N%; "elements sorted in"; (FS-SS)/18; "seconds"

# **Dialing for Dollars**

### Writing Your Own Telecommunications Program

Donald B. Trivette

Not only does this article give you a program for retrieving Dow Jones stock quotes—it also explains in detail how the program works so you can modify it for your own purposes. As written, the program automatically calls up the Dow Jones News/Retrieval Service, logs on with your password, retrieves any current quotes you want, displays or prints them out, and logs off—in as little as 22 seconds. It's written for the Hayes Smartmodem 1200, but can be made to work with other modems. The program requires BASICA on the PC or Cartridge BASIC on the PCjr, a modem, and an account with Dow Jones/News Retrieval.

Do you have an urge to call your stockbroker a dozen times a day to check the value of your stocks? Would you like to have your very own ticker-tape machine? If your portfolio exceeds a million dollars, your broker may take your calls or you may indeed have a quote machine on your desk—but the rest of us can't get quotations so easily. Now there's a way to get quotes as often as you want them—maybe even free.

You probably know that Dow Jones & Company offers a stock-market quotation service that can be accessed with an IBM PC or PCjr. To tap into this service, you can use a general-purpose communications program such as PC-Talk or Crosstalk. Or you can use a program specifically designed for the Dow Jones News/Retrieval Service, such as the Dow Jones Market Analyzer. Or you can write your own. (Note: To use any of these programs, your PC must be equipped with a modem and you must have an account with Dow Jones.)

One advantage to using a custom program is that it can be tailored to do *exactly* what you want—with maximum speed and minimum input. Another is that you can embed such a program in a larger program, perhaps your own portfolio analysis software. Even if you have no interest in the stock

market, read on; you can modify the program we're going to build to access any database. This program is specifically written for a Hayes Smartmodem 1200 and the Tymnet telephone network—if you have another brand of modem or use a different telephone network, such as Telenet, you'll need to make alterations.

### **Defining the Task**

The first thing to decide when writing a computer program is what the program is to do. Simple enough. We want our program, named "Quotes," to call up the Dow Jones News/Retrieval Service, to enter password and other log-on parameters, and then, using symbols stored in a file, to collect market quotations. We want to save these quotations for display and printing. And, because Dow Jones charges by the minute, we want to get in and out as quickly as possible.

Before we can get the computer to do this automatically, we've got to do it manually. We've got to know what the input/output sequence looks like. To do this, let's use the *PC-Talk* program (Freeware, P.O. Box 862, Tiburon, CA 94920; \$35) to establish the connection and retrieve five quotes. At the same time *PC-Talk* is sending and receiving data, it can write everything to a disk file. Once the session is over, the disk file contains a complete record of what took place. For an idea of what a session looks like, see Figure 1 (my entries in bold).

Let's examine the output. The first line is the command to the Hayes Smartmodem 1200 to dial a telephone number. The AT alerts the modem that what follows is a modem command, not something to send over the phone line. The D says *dial* and the T says *use Touch-Tone dialing*; the number is the telephone number for Tymnet in my area of North Carolina. (The Hayes manual explains the modem commands in detail.)

The next line is sent back to the computer by the modem. It tells the computer that a connection has been made with another computer—in this case the Tymnet computer. If the number had been busy, the modem would have sent a NO CARRIER message to the computer.

The third line looks like garbage, and is. It really says, *Enter your terminal identifier*. If you're running the modem at a slower speed, 300 baud, it will say that, but if you're running at 1200 baud, it says, xxx < xx... Not everything I typed was captured in the disk file. In response to this line, I typed A—the terminal type for the PC.

Figure 1. PC-Talk Session

```
ATDT343-0770
CONNECT
xx'xxx<'xx''xx<'x'x<xxx'xxxxxxx'x<xxxxx<'xx
-3625-022-
please log in: dow2;
tc> host is online
WHAT SERVICE PLEASE????
ENTER PASSWORD
WWWWWWWWWWWWW
MMMMMMMMMMMMM
000000000000000000
 password
 DOW JONES NEWS/RETRIEVAL
  COPYRIGHT (C) 1984
 DOW JONES & COMPANY, INC.
  ALL RIGHTS RESERVED.
SPINKS KEEPS LIGHT HEAVYWEIGHT
TITLE IN UNANIMOUS DECISION,
SEE //SPORTS. SEE //MENU
FOR A DATA-BASE LIST.
ENTER QUERY
 //cq
CURRENT DAY QUOTES BEING ACCESSED
ENTER QUERY
 ibm ge ek abg abcde
DOW JONES STOCK QUOTE REPORTER SERVICE.
STOCK QUOTES DELAYED OVER 15 MINUTES
*=CLOSE PRICE ADJUSTED FOR EX-DIVIDEND
STOCK BID
                ASKED
                                  LOW
       CLOSE
                OPEN
                         HIGH
                                           LAST
                                                    VOL(100's)
IBM
      109 1/4 109 1/4
                         111 1/2
                                  109 1/4 111 1/4
                                                    10432
GE
                                  52 3/8
       52
                52 3/8
                         54
                                           53 3/4
                                                    10235
       *66 3/4
                         69
                                  673/4
EK
                67 7/8
                                           69
                                                    4188
ABG 11
                         11
                                  10 3/4
                                           11
                                                    60
ABCDE STOCK SYMBOL IN ERROR
LOG ON: 19 38 LOG OFF: 19 39 EASTERN TIME FEBRUARY 25, 1984
tc> dropped by host system
please log in:
```

The next line is a Tymnet identification number, followed by a prompt for the computer we want to talk to. We want to talk to the DOW2;; computer. Notice that we entered *two* semicolons, although only one was printed. It's important to use two semicolons.

More lines from the computer follow, and finally a prompt asking what service. We want the Dow Jones News Service (djns). More lines and a prompt for the password. I entered mine (it doesn't print)—you enter yours.

Still more output from DJ, including a sports update. These messages end with ENTER QUERY and a special character, ASCII 30, that's not visible on a printout. (Look in Appendix G of your BASIC manual, and you'll find that ASCII 30 moves the cursor up.) This is the character that DJ sends when it expects input from us. The same character is in the password prompt. Later we'll see how to detect and use this unprintable character.

### **Retrieving the Quotes**

The query we want to make is about *current quotes*. DJ expects queries to begin with two slashes (//), thus our response of //cq. (I typed my input in lowercase, but DJ understands

either upper- or lowercase.)

We get more trash (really heading information), and then the invisible ASCII 30 character again. Here we enter the stock symbols for which we want quotations. As many as five symbols may be given per line; symbols for stock options, bonds, and so on are preceded with a special mark. (The *Dow Jones News/Retrieval Fact Finder* book you get when you open a DJ account lists stock symbols as well as these special marks.)

The DJ computer responds to our request with two lines of headings and five lines of quotation information—one line for each stock symbol. Notice the response for an invalid symbol (STOCK SYMBOL IN ERROR).

After sending the answer to our query, DJ sends ASCII 30 again. It's ready for more symbols. We are ready to disconnect. To disconnect from the DJ computer, enter *disc*. There's no stock with this symbol.

The *disc* command drops the connection with the DJ computer but not the connection to Tymnet. No matter—our charges cease with the *dropped by host system* message—Tymnet drops when we hang up the telephone.

**Interpreter or Compiler?** 

Now we can write the Quotes program. First, load Advanced BASIC with a 2K communications buffer—at the DOS prompt type BASICA/C:2048. The PCjr requires Cartridge BASIC. Type in the program exactly as it appears at the end of this article, but don't type blindly. At least read through the description below before running Quotes. (You will need to modify line

930 if you've changed the default switch setting on the Hayes modem.)

First, the program clears the screen, turns off the function keys display, and closes any files which might be open. Next we need to find out if Quotes is being run in compiler BASIC or the regular interpreter BASIC. Why? What difference does it make?

With interpreter BASIC (which includes Disk BASIC and BASICA on the PC and Cartridge BASIC on the PCjr), we'll create a scroll window at the bottom of the screen to display the data going back and forth to the Dow Jones computer. There's a handy POKE which does that. But if Quotes is compiled, the scroll window won't work. Thus, in compiled form we'll skip over statements which display the communications traffic.

There's a trick to determine which form of BASIC a program is running under: dimension an array, set one element equal to the value 1, and then use the ERASE instruction. Compiler BASIC doesn't support the ERASE instruction and thus won't change the value of MODE(1). Interpreter BASIC will ERASE the array, and MODE(1) will have a value 0. (Line 230 will be flagged as an error when Quotes is compiled. Ignore that. It won't affect linking or running the program.)

Next, the program establishes the ON conditions (lines 270–290). We're going to set aside the F1 key to abort the program in an emergency. The subroutine at line 2590 disconnects the modem and hangs up the telephone. The ON ERROR statement tells the program how to respond when it encounters an error.

Line 340 sets the entire screen as the scroll window when the program is running under interpreter BASIC. POKE 91,1 tells BASIC that the scroll window begins at line 1 on the screen; POKE 92,25 tells BASIC the window ends with line 25. (The default settings are lines 1 and 24. Line 25 doesn't usually scroll.)

Creating a Symbol File

Line 380 dimensions two arrays. The Q\$ array holds the lines we want to transmit to DJ; the QL\$ array stores what we receive from DJ. There are several reasons to save this information rather than just printing it as it comes across the telephone line.

First, since telephone connections are prone to noise, the information may contain characters we'll want to remove. Second, we don't want to display the heading information continuously. Third, we'd like the display to be single-spaced. And fourth, we want to be able to read and review the quotes while the computer is offline, rather than online where it's costing us money.

The next step (line 490) is to read the name of the symbol file. The symbol file is where we'll store the symbols for the stocks we want quoted. It's also where we'll store the Tymnet telephone number and the DJ password. You can build the symbol file with any text editor or word processing program which creates files in ASCII form. You can even use the EDLIN program which came on your PC-DOS disk.

The symbol file for the session shown above is

343-0770 your-dj-password ibm ge ek abg abcde disc

The symbols *ibm ge ek abg* are for IBM, General Electric, Eastman Kodak, and American Ship Building; *abcde* is one I made up to demonstrate what happens when you enter an incorrect stock symbol. Your file may have as many lines of symbols as you want—just remember to put no more than five to a line.

If you don't enter the name of a file—that is, if you just press the Enter key—the program defaults to a file named DJSYMBOL on the A: disk drive. You may want several symbol files: one for stocks you own, another for stocks you're thinking of buying, and perhaps a third for those you've sold (if you're a masochist). How you create, and what you put in your symbol files, is up to you.

Once Quotes knows the name of the symbol file, it reads the contents into the Q\$ array (lines 600–660). Should the file not exist, the error routine at line 1800 will beep, print a message, and let you reenter the filename.

Interfacing with the Modem

Now we have to find out what transmission speed to use. If your modem has only one speed, you have only one choice.

The Hayes 1200 can operate at either 300 or 1200 baud—1200 is the speed of choice. The faster we get in and out, the less we'll be charged by Dow Jones.

At this point (line 850), if the program is running under interpreter BASIC, it shrinks the scroll window to the bottom five lines of the display. The top is at line 20, the bottom at line 25. The LOCATE statement puts the cursor in that window; it isn't done automatically.

In line 920, we pop up to line 12 to print a message and then back down to line 20, back down to the window. Now we're ready to talk to the modem.

Line 930 is straight out of the BASIC manual for opening a communications file. COM1: is the port where the modem is connected. When we want to send something to DJ, we'll print it to file #1; when we want to read something from DJ, we'll read from file #1. A COM file allows both input and output at the same time. (Line 930 is correct for the Hayes Smartmodem 1200 with the configuration switches in the factory setting; if switch 6 is up, as required by communications software such as *Crosstalk*, then use *E*,7,1,CS,DS as the parameters in the OPEN statement.)

The first thing the program sends to the modem is the telephone number we want it to dial (line 1040). If you want, you can change the value of S7, the number of seconds the modem allows for the remote computer to answer the telephone, and S12, the guard time. (More about S12 later.)

Once we've told the modem to dial the telephone number, we sit back and wait for an answer—that is, an answer from the modem. It'll tell us when it has a computer on the other end of the line. Lines 1130–1170 take care of that.

### **Sending and Receiving Characters**

Communications with the Dow Jones database, as with most databases, is by an *asynchronous* technique. Each character is sent and received separately, one at a time; each character is a discrete unit.

BASIC has a built-in function which makes asynchronous communications a little easier. The LOC function returns the number of characters in the communications buffer (file). If LOC is zero, there's nothing in the file to read. The EOF (end-of-file) function can also be used with communications files. If

EOF is true, the buffer is empty. But remember, characters are sent to or received from the buffer one at a time. Telecommunications via modem is much slower than the computer itself. The buffer may be empty only because the modem hasn't had time to put something in it. The EOF and LOC tests don't give

us the true picture.

Lines 1130–1170 form a continuous loop. If the buffer is not empty, line 1140 reads what characters *are* there into the variable A\$. Then those characters are tacked onto the rightmost 20 characters stored in the variable Z\$. In other words, Z\$ is going to be the last 20 characters which came from the buffer, plus the value of A\$. Why not just form one character string of all the characters which come from the buffer—something like Z\$=Z\$+A\$, for example? Because BASIC limits strings to 255 characters, so we'd soon get an error if we did it that way. Besides, we need only enough characters to look for the CONNECT message.

In lines 1150 and 1160, the program tests the Z\$ character string for one of two values: CONNECT or NO CARRIER. The Hayes modem will put one or the other of those strings in the buffer. If neither is present, the program must go back and

continue to read from the buffer.

Notice *how* it reads from the buffer: an INPUT\$ statement. The statement INPUT\$(n,f) has two parameters: n is the *number* of characters it is to read, and f is the *file number* from which it is to read. For communications files, we usually want to read everything that's in the file. We want to read LOC(1) characters from file #1. Thus INPUT\$ (LOC(1),#1).

Once we've got a connection, the program jumps to line 1240 and begins reading the buffer again. Every time the buffer comes up empty (EOF), the program tests to see whether it has something we recognize.

**Looking for Cues** 

The first thing we want is either *Enter terminal identifier* (when communicating at 300 baud) or the string of garbage as seen in the sample session (1200 baud). Line 1290 identifies the slower-speed message and responds by transmitting an *A*. Similarly, line 1300 catches the faster-speed message and also sends an *A* to the Tymnet computer.

In line 1300, why does the program look for CHR\$(0) followed by CHR\$(120)? That's where the sample session stored

in the disk file comes in handy. What we see displayed on the screen isn't necessarily everything that comes across the telephone line. By using the *Norton Utility* program (The Norton Utilities, 2210 Wilshire Boulevard, Santa Monica, CA 90403; \$80) to display the sample session in hexadecimal format, we can see that the null character—CHR\$(0)—precedes the lowercase x—CHR\$(120)—in the garbage.

Notice how line 1290 looks for the fragment *ifier* instead of the whole message *Enter terminal identifier*. Noise on the telephone line might garble part of the longer message; it might read *Enter ter*{*minal identifier*. This, of course, wouldn't match what we're looking for. The program must keep the test string as short as possible, yet long enough to identify a unique message.

Some of the PRINT #1 statements (lines 1290–1350) end with a semicolon, others don't. When the semicolon is present, BASIC does *not* send a carriage return along with the data. It's important *not* to send a carriage return for some lines, and important *to* send it with others. The program must duplicate what you would enter on the keyboard.

Lines 1310 and 1320 are almost identical. They transmit the same data, and there's a good reason for this. If for some reason the Tymnet computer doesn't understand our response to please log in:—because of telephone line noise, for example—it will respond with enter user name:. At that point we can recover by sending DOW2;;. In fact, this section of code (lines 1240–1370) will be executed until the program receives the message CURRENT DAY QUOTES BEING ACCESSED. When writing a communications program, it's best to design it to retry until it gets an answer it understands.

### Retrieving the Quotations

The next section of the program (lines 1560–1730) gets a little tricky. The idea is to read characters from the buffer and stick them onto a line we're saving, OL\$(L), until the program encounters a linefeed character, CHR\$(10). Then, if the line is less than 15 characters long, the program throws it away. (The lines we want from the DJ computer are always longer than 15 characters.)

If the characters from the buffer contain a CHR\$(30), the program sets a switch to remember it's ready to transmit again. Once it processes the data stored in A\$, it checks to see

whether the buffer is still empty (line 1680). If something has come in while the program was doing all this, it must go back and read it. Otherwise, it drops down to line 1690 to see if it's transmitted all our symbols, or down to line 1700 to see if it's ready to transmit another one.

Lines 1560–1730 are the heart of the program. They are the statements which send out the stock quote symbols and save the answers. I'd like for you to think these statements worked the very first time I tried them. They didn't. It took hours of trial and error to come up with this code. I learned that before transmitting, you must always test to be absolutely sure the buffer is empty. Otherwise, the output gets mixed with the input. (Why didn't the manuals say that?)

Hanging Up the Phone

The rest of the program is easy. Once it's finished collecting quotes, it hangs up the telephone. That's done in the subroutine beginning at line 2590. The program sends CHR\$(19) to the DJ computer. That's the X-off code, the code that instructs the remote computer to send us no more data. (Frankly, I never could tell if this statement was doing anything, but it didn't seem to hurt. I left it in the program just to be safe.)

Next, the program must send a hang-up message to the Hayes modem. The Hayes manual explains that to get the modem to switch to local mode and hang up, you pause for at least one second, then send three plus signs (+++), but no carriage return, then pause again for one second, and finally send  $AT\ H0$ . The pauses are called the *guard time*. The idea is that you wouldn't normally be sending three pluses to the remote computer, and certainly not with a second's pause immediately before and after. (If you would, there are ways to change both the plus character and the guard time to something else.)

The program can use two SOUND statements, out of human hearing range, to time the necessary pause. By telling the computer to make a sound at 32767 hertz for 20 clock ticks—the computer's clock ticks 18.2 times per second—and putting another SOUND statement immediately after this one, we insure a pause of slightly over one second. The second SOUND statement causes BASIC to wait until the first one is finished before execution continues. The program keeps reading from the buffer and looking for the OK message from the modem

(line 2710) which indicates the hang-up command was successful.

Unfortunately, this method didn't always hang up the phone. But I found that reducing the guard time to zero seconds—that's what the S12 parameter in line 1040 does—always disconnects the modem properly. This, of course, could cause problems if we have a reason to send three pluses to Dow Jones. But we don't.

The remainder of this subroutine either ENDs the program (line 2750)—if you press the F1 key—or returns to line 1920, where the program is ready to display the stock quotes it captured.

### Displaying the Information

Before the program can display what it's saved in the QL\$ array, it must do a little editing. Lines 1970–2070 take care of that. We want to throw away any line containing the words *LOG OFF*; also any line with CHR\$(30), VOL(100), and any line less than 15 characters, unless it also has the word *ERROR* (this preserves the SYMBOL IN ERROR message).

Lines 2040–2070 are terribly slow, but unless your telephone connections are of pristine quality, you'll want to keep them in the program. These statements examine every character in the display lines and remove any that are less than ASCII value 32 or greater than ASCII 90. The only valid characters the DJ computer will send are between ASCII 32 and 90. Everything else must be garbage. If you compile Quotes, these statements won't cause a delay.

Figure 2. Stock Quote

Dow Jones Report 02-25-1984 20:03:40						
Symbol	Close	Open	High	Low	Last	Volume
IBM	109 1/4	109 1/4	111 1/2	109 1/4	111 1/4	10432
GE	52	52 3/8	54	52 3/8	53 3/4	10235
EK	*66 3/4	67 7/8	69	67 3/4	69	4188
ABG	11	11	11	10 3/4	11	60
ABCDE	BCDE STOCK SYMBOL IN ERROR					

### **Notes for Compiler BASIC**

To compile Quotes, you must have the IBM BASIC compiler. Here are the steps:

- 1. Save Quotes with the ASCII option (type SAVE "QUOTES.BAS",A).
- 2. Run the compiler with the following compiler options: X/O/V/C:2048. The C:2048 option tells the compiler to generate code with a 2K communications buffer.
- 3. In the link step, you must include the IBM COM library. To do this, use LINK QUOTES.OBJ+IBMCOM.OBJ.
- 4. The resulting program will be named QUOTES.EXE. It will run considerably faster and without first having to load BASIC.

### Free Stock Quotations?

Now some information about speed, cost, and the Dow Jones News/Retrieval Service. Dow Jones charges 90 cents per minute during prime hours (6:00 a.m. to 6:01 p.m. your local time, Monday through Friday), and 20 cents per minute at other hours. Using 1200 baud or 2400 baud costs 2.2 times these charges.

Quotes takes only 22 seconds (at 1200 baud) to retrieve 14 common stock quotations. Since Dow Jones *rounds* to the nearest minute, it will cost you nothing if you get in and out in less than 30 seconds. (Expect this to change if DJ is flooded with 29-second connections.)

Dow Jones seems to be busiest between 6:00 p.m. and 8:00 p.m. weeknights. During those hours, the hundreds of other moguls retrieving quotes may double, even triple, the time it takes Quotes to do its stuff.

Dow Jones offers lots of other services—historical quotes, news reports, MCI mail, even movie reviews—but Quotes works only on the Current Quotes (delayed 15 minutes) database. To open a Dow Jones account, call 1-800-257-5114 (1-609-452-1511 in New Jersey, Alaska, and Canada), or write to Dow Jones & Company, P.O. Box 300, Princeton, NJ, 08540.

### Quotes

For error-free program entry, be sure to use "The Automatic Proofreader," Appendix B.

AF 140 CLS

LE 150 KEY OFF

PC 160 CLOSE

```
FJ 17Ø REM ----
HB 180 REM Check for Compiler BASIC
DO 190 REM mode(1)=0 interpreter, mode(1)=1 compi
      1er
EN 200 REM -----
9J 21Ø DIM MODE(1)
HD 22Ø MODE(1)=1
06 23Ø ERASE MODE
EE 240 REM -----
#M 250 REM Establish ON conditions
EI 260 REM -----
PD 270 ON ERROR GOTO 1780
M 28Ø ON KEY(1) GOSUB 259Ø
NF 290 KEY(1) ON
EN 300 REM -----
0A 310 REM If interpreter BASIC, make scroll
HE 320 REM Window all 25 lines
ED 33Ø REM -----
EH 340 IF MODE(1)=0 THEN DEF SEG : POKE 91,1:POKE
      92,25
EH 350 REM -----
ME 360 REM Set up storage arrays and print headin
      qs
FL 370 REM -----
8N 38Ø DIM Q$(100),QL$(200) 'Q$ stores input, QL
      $ stores output
CE 390 FILOPN=0
                            'Com file is closed
OK 400 PRINT SPC(26) "Dow Jones Quotations"
IF 410 PRINT
IH 420 PRINT
ND 430 PRINT "Press F1 to manually disconnect at
      any time":PRINT
JL 44Ø PRINT
E 45Ø REM -----
40 460 REM Now get name of symbol file from keybo
      ard
FM 470 REM If no name entered, default to "DJSYMB
      OL "
FO 480 REM -----
JP 490 LOCATE 7,1:PRINT SPACE$ (70)::LOCATE 7,1:LI
      NE INPUT "Enter Symbol File: ";A$
CM 500 IF AS="" THEN FILENS="DJSYMBOL":LOCATE 7,1
      :PRINT SPACE$ (70)::LOCATE 7.1:PRINT "Enter
       Symbol File: ";FILEN$ ELSE FILEN$=A$
E9 51Ø REM -----
SH 520 REM Format of input (symbol) file
MK 530 REM TYMNET telephone #
6M 54Ø REM DJ password
FH 55Ø REM
          Quote lines -- 5 symbols/line
EK 560 REM DISC
```

```
46 57Ø REM
86 580 REM Read file into @$ array
FB 59Ø REM -----
#8 600 OPEN FILENS FOR INPUT AS #3
01 61Ø IF EOF(3) THEN 66Ø
JF 62Ø I=I+1
FD 630 LINE INPUT #3,A$
NJ 640 Q$(I)=A$
F! 45Ø GOTO 41Ø
PH 660 CLOSE
FO 670 REM -----
NI 680 REM Save number of symbol lines as KMAX
ME 690 REM and ask user which speed
ER 700 REM ----
HE 710 KMAX=I
₩ 72Ø PRINT
MA 730 PRINT "Enter: F for 1200"
NG 740 PRINT "
                   S for 300"
JA 75Ø PRINT
IC 760 A$=INKEY$:IF A$="" THEN 760 'wait for ans
HN 770 IF A$="S" OR A$="s" THEN BAUD$="300":60TO
JJ 780 IF A$="F" OR A$="f" THEN BAUD$="1200":GOTO
       850
C! 790 BEEF: GOTO 760 'Neither choice correct,
      try again
EC 800 REM ----
NF 810 REM If interpreter, print a heading and
IE 820 REM make screen lines 20-25 the scroll win
      dow:
86 830 REM move the cursor there.
FK 84Ø REM -----
OK 850 IF MODE(1)=0 THEN LOCATE 19,1:PRINT STRING
      $(8Ø, "-");:LOCATE 19,29:PRINT"Communicatio
      ns Window";:DEF SEG: POKE 91,20:POKE 92,25
      :LOCATE 20
FO 86Ø REM -----
M 870 REM Print message. Open the COM1: file (th
      e modem)
IL 880 REM for the correct speed, even parity, 7
      bits,
AC 890 REM and 1 stop bit (include DS if SW6 is u
      p)
ED 900 REM ----
FL 910 NUMBER$=Q$(1) Telephone number
81 920 LOCATE 12,1:PRINT SPACE$(70)::LOCATE 12,1:
      PRINT "Dialing Dow Jones: "; NUMBER$: LOCATE
KE 930 CLOSE: OPEN "COM1: "+BAUD$+", E, 7, 1, CS" AS #1
```

```
FL 940 RFM -----
M 950 REM Remember that COM1 is now open (FILOPN
      =1).
BD 960 REM Initialize Is and send (PRINT #1) the
      modem
NG 970 REM the telephone number. Tell the modem t
      o wait
PE 980 REM 20 seconds for an answer (S7) and that
       the
NL 990 REM "quard time" for the escape characters
       is Ø (S12).
HD 1000 REM After the number is dialed, the modem
        is to
州 1010 REM go "on-line" (D).
HD 1020 REM -----
NA 1030 FILOPN=1: Z$=""
JL 1040 PRINT #1, "AT D"+NUMBER$+"; $7=20 $12=0 0"
HH 1050 REM -----
06 1040 REM Now begin reading the COM1: buffer (f
       ile) to find
HB 1070 REM out whether our modem actually got co
       nnected to another
HA 1080 REM modem.
66 1090 REM If yes (CONNECT), then print message
       and goto next step.
NG 1100 REM If no (NO CARRIER), then print "BUSY"
        and find out
6A 1110 REM if user wants program to keep dialing
HA 1120 REM -----
```

- NG 113Ø IF EOF(1) THEN 115Ø
- LK 114Ø IF LOC(1)>Ø THEN A\$=INPUT\$(LOC(1),#1):Z\$= RIGHT\$(Z\$,2Ø)+A\$
- B 115Ø IF INSTR(Z\$,"NO CARRIER")<>Ø THEN LOCATE
  12,1:PRINT SPACE\$(7Ø);:LOCATE 12,1:PRINT"
  B U S Y":LOCATE 2Ø:GOSUB 23ØØ
- MC 1160 IF INSTR(Z\$,"CONNECT")<>0 THEN LOCATE 12, 1:PRINT SPACE\$(70);:LOCATE 12,1:PRINT "Co nnected to Dow Jones":Z\$="":LOCATE 20:GOT D 1240
- LA 1170 GOTO 1130
- IC 1180 REM -----
- NJ 1190 REM Next, we continually read the COM1: buffer
- DA 1200 REM while looking for specific things. When we find
- CN 121Ø REM something we recognize, we send the a ppropriate
- 66 1220 REM response.
- HF 1230 REM ----

```
FN 1240 IF EDF(1) THEN 1290
06 1250
          IF LOC(1) >Ø THEN A$=INPUT$(LOC(1),#1)
JO 1260
          IF MODE (1) = Ø THEN PRINT A$;
GD 127Ø
         Z$=RIGHT$ (Z$, 2Ø) +A$
         A$=""
@J 128Ø
          IF INSTR(Z$, "ifier")<>Ø THEN PRINT #1,
JH 129Ø
       "A":: 7$=""
OP 1300
          IF INSTR(Z$,CHR$(Ø)+CHR$(12Ø))<>Ø THEN
       PRINT #1, "A"; : Z$=""
KG 1310
          IF INSTR(Z$, "in:")<>Ø THEN PRINT #1, "DO
       W2;;";:Z$=""
DH 132Ø
          IF INSTR(Z$, "me:")<>Ø THEN PRINT #1, "DO
       W2;;";:Z$=""
          IF INSTR(Z$,"???")<>Ø THEN PRINT #1,"DJ
KM 133Ø
       NS" : Z$=""
HD 1340
          IF INSTR(Z$, "aaaaaaa") <> Ø THEN PRINT #1,
       Q$(2):Z$=""
LK 135Ø
         IF INSTR(Z$, "UERY") <> Ø THEN PRINT #1, "/
       /CQ" : Z$="":LOCATE 12,1:PRINT SPACE$ (70);
       :LOCATE 12,1:PRINT "Pass-word accepted.":
       LOCATE 20
          IF INSTR(Z$, "ESSED") <> Ø THEN Z$="":TIME
BJ 1360
       B$=TIME$:GOTO 1450
LN 1370
       GOTO 124Ø
IG 138Ø REM -----
NN 1390 REM Now we have passed from the TYMNET co
       mouter
FC 1400 REM to the DJ computer, and DJ has accept
       ed our
P 1410 REM password. We are ready to send the fi
       rst line
ME 1420 REM of symbols. But first, we blank out t
       he array
ME 1430 REM where we'll store the DJ responses.
IN 1440 REM -----
@L 1450 LOCATE 12,1:PRINT SPACE$(70);:LOCATE 12,1
       :PRINT "Collecting quotes.":LOCATE 20,1
01 1460 FOR I=1 TO 200
DP 147Ø QL$(I)=""
HP 148Ø NEXT I
JL 149Ø REM ----
IK 1500 REM We send a line of symbols -- Q$(L) --
NI 1510 REM wait for an answer. We save the answe
       r in
PL 1520 REM QL$(L). We look for the ASCII charact
AD 1530 REM CHR$(30) -- that tells us DJ is ready
```

PA 1540 REM us to send another line.

```
IB 155Ø REM -----
J6 1560 L=1:K=3:CMSW=1
HM 1570 IF LOC(1)>0 THEN A$=INPUT$(LOC(1),#1)
KK 158Ø IF MODE(1)=Ø THEN PRINT A$:
        IF INSTR(A$, CHR$(3Ø))<>Ø THEN CMSW=1
EL 1590
80 1600 LF=INSTR(A$, CHR$(10))
KA 1610
           IF LF=Ø THEN 166Ø
01 1620
           QL$(L)=QL$(L)+MID$(A$,1,LF-1)
            IF LEN(QL$(L))<15 THEN QL$(L)="" EL
LK 1630
       SE L=L+1
DK 1640
            A$=MID$ (A$, LF+1)
LJ 1650
        GOTO 1600
MA 1660 QL$(L)=QL$(L)+A$
JC 167Ø A$=""
HB 168Ø IF LOC(1)=Ø THEN 169Ø ELSE 157Ø
CP 1690
            IF K>KMAX THEN GOTO 1850
IO 1700
            IF CMSW=Ø THEN 172Ø
HD 1710 IF LOC(1)=0 THEN PRINT #1,Q$(K):LOCATE 12
       ,1:PRINT SPACE$(70);:LOCATE 12,1:PRINT Q$
       (K):K=K+1:CMSW=0:LOCATE 20 ELSE GOTO 157
IF 172Ø A$=""
BE 173Ø GOTO 157Ø
IC 1740 REM -----
MI 1760 REM Error-handling routine
JO 1780 REM -----
PE 1790 IF ERR= 57 THEN RESUME
N 1800 IF ERR=53 AND ERL=600 THEN LOCATE 7:PRINT
        "ERROR: Symbol file ";FILEN$;" not found
                 ":BEEP:SOUND 32767,60:SOUND 327
       67,1:RESUME 49Ø
PJ 1810 PRINT "***"; ERR, ERL; "***": IF MODE(1)=0 TH
       EN POKE 91,1:POKE 92,25:STOP ELSE STOP
10 182Ø REM -----
MH 1830 REM Finished. Disconnect from modem.
IE 184Ø REM -----
P 1850 TIMEES=TIMES
JD 1860 GOSUB 2610
JN 187Ø REM ----
NA 1880 REM Now we're ready to "edit" and print w
       hat
KG 1890 REM we got from DJ. If necessary, we set
# 1900 REM the scroll window to fill the whole s
       creen.
IN 1910 REM -----
FP 1920 CLS: IF MODE(1)=0 THEN POKE 91,1:POKE 92,2
       5: CLS
DE 1930 PRINT "Symbol
                        Close
                                   Open
                                             High
                       Last
                                  Volume"
             LOW
LC 1940 PRINT STRING$(67,"-")
```

```
CM 195Ø LI=3
₩ 196Ø FOR I=1 TO L
# 1970 IF INSTR(QL$(I), "ERROR")<>0 THEN GOTO 204
EF 1980 IF INSTR(QL$(I), "LOG OFF") <>0 THEN QL$(I)
       ="":GOTO 214Ø
FB 1990 IF INSTR(QL$(I), CHR$(30))<>0 THEN QL$(I)=
       "":GOTO 214Ø
£P 2000 IF INSTR(QL$(I), "VOL(100") <>0 THEN QL$(I)
       ="":GOTO 214Ø
HC 2010 IF INSTR(QL$(I), "STOCK ")<>0 THEN QL$(I)=
       "": SOTO 2140
D 2020 IF INSTR(QL$(I). "ADJUSTED") <>0 THEN QL$(I
       )="":60TO 214Ø
HC 2030 IF LEN(QL$(I))<15 THEN QL$(I)="":GOTO 214
FK 2040 FOR J=1 TO LEN(QL$(I))
EE 2050 Z$=MID$(QL$(I),J,1)
KD 2060 IF Z$<CHR$(32) OR Z$>CHR$(90) THEN QL$(I)
       =MID$(QL$(I),1,J-1)+MID$(QL$(I),J+1):GOTO
        2040
HA 2070 NEXT J
KH 2080 PRINT QL$(I)
PH 2090 LI=LI+1
61 2100 IF LI<20 THEN 2140
80 2110 PRINT SPC(10)"--more--"
NA 2120 ANS$=INKEY$: IF ANS$="" THEN 2120
FJ 213Ø CLS:LI=1
GO 214Ø NEXT I
# 215Ø PRINT:PRINT "Begin: ";TIMEB$;" End: ";
       TIMEE$
IN 2160 REM -----
IJ 2170 REM Ask if user wants printed copy.
@M 218Ø REM If yes, then print; if not, end.
IS 219Ø REM -----
PI 2200 PRINT:PRINT "Do you want a printed copy (
       Y/N):"
N 2210 A$=INKEY$: IF A$="" THEN 2210
AE 2220 IF A$="Y" OR A$="y" THEN GOSUB 2420
LC 223Ø CLOSE: END
IJ 2240 REM -----
JE 2250 REM This routine is for a BUSY number.
MO 2260 REM First, disconnect from modem and
IE 2270 REM then find out if user wants the
LA 2280 REM PC to keep trying.
JI 229Ø REM -----
19 2300 ERC=0
18 231Ø B$=""
10 232Ø GDSUB 261Ø
```

```
0A 233Ø IF CONSW=1 THEN LOCATE 12,1:PRINT SPACE$(
       70);:LOCATE 12,1:PRINT "Re-dialing...":LO
       CATE 20: RETURN 930
DI 234Ø LOCATE 12,1:PRINT SPACE$(7Ø);:LOCATE 12,1
       :PRINT "Do you want auto re-dial (Y/N):":
       LOCATE 20
PP 2350 B$=INKEY$:IF B$="" THEN 2350
DN 2360 IF B$="Y" OR B$="y" THEN CONSW=1:LOCATE 1
       2,1:PRINT SPACE$(70);:LOCATE 12,1:PRINT"R
       e-dialing...":LOCATE 20:RETURN 930
en 2370 LOCATE 12,1:NORET=1:RETURN 2750
IH 2380 REM -----
IP 2390 REM This routine prints a
DN 2400 REM report on the printer.
HE 2410 REM -----
MP 2420 WIDTH "lot1:".80
©D 243Ø LPRINT SPC(32) "Dow Jones Report"
MC 244Ø LPRINT SPC(3Ø) DATES;"
                                 ": TIME$
6K 245Ø LPRINT
GJ 2460 LPRINT SPC(10) "Symbol
                                 Close
                                           Open
           High
                     LOW
                                           Volume"
                                Last
IH 2470 LPRINT SPC(10)STRING$(67."-")
ND 2480 FOR J=1 TO L
H 2490 IF QL$(J)="" THEN 2510
JK 2500 LPRINT SPC(10) QL$(J)
ek 2510 NEXT
EM 2520 LPRINT CHR$ (12)
JP 2530 RETURN
IP 254Ø REM -----
HH 2550 REM This routine disconnects the modem.
MJ 2560 REM We need to know if we got here by pre
       ssing
KI 2570 REM the F1 key or by a regular disconnect
JL 258Ø REM -----
JA 2590 REM Arrived via F1
ML 2600 NORET=1
JI 2610 REM Arrived via disconnect
CH 2620 IF FILOPN=0 THEN RETURN
DK 2630 REM local mode
KC 264Ø PRINT #1, CHR$(19);
JK 265Ø SOUND 32767, 20: SOUND 32767, 1
JP 2660 PRINT #1,"+++";:BEEP
JA 2670 SOUND 32767, 20: SOUND 32767, 1
# 268Ø PRINT #1, "AT HØ"
KJ 269Ø A$=""
CF 2700 IF LOC(1)>0 THEN A$=A$+INPUT$(LOC(1),#1)
HF 2710 IF INSTR(A$, "OK") <>0 THEN 2730
HA 2720 IF LOC(1)>0 THEN 2700 ELSE 2650
```

BP 273Ø CLOSE #1:FILOPN=Ø:ZZZ\$=A\$

# All About Programming

- EB 2740 LOCATE 12,1:PRINT SPACE\$(70);:LOCATE 12,1 :PRINT "Line disconnected.":PRINT
- IH 275Ø IF NORET=1 THEN PRINT "Program terminated
   ."+SPACE\$(14):GOTO 276Ø ELSE RETURN
- MD 2760 IF MODE(1)=0 THEN POKE 91,1:POKE 92,25:EN D ELSE END
- HG 2770 REM Last line in program QUOTES

# Advanced Function Key Techniques

Peter F. Nicholson, Jr.

Restoring original key definitions, extending definitions for certain keys beyond the default limits, and saving definitions to disk for later use are among the techniques covered in this revealing article. For the IBM PC and PCjr and most compatibles.

Anyone who has ever redefined the function keys in an IBM BASIC program probably has wondered why there's no command to restore the keys' original definitions when the program ends. Usually, you end up disabling them or redefining them again to their default values. But there's an alternative, and the secret lies within something called the *soft key buffer*. Locating and examining this buffer can yield some interesting results.

Finding the buffer is easy if you have an IBM PC, XT, or PCjr. It starts at memory location 1619 in the default memory segment. But this is not necessarily true if you have an IBM-compatible. Therefore, if you're using a compatible, you should run Program 1, which attempts to locate the soft key buffer for you. When you find it, you should alter the buffer address (1619) in the IBM programs before running them on your compatible. The appropriate lines are indicated in RE-Mark statements within each program.

# Saving Key Definitions

The soft key buffer is just a section of memory which stores the definitions for the function keys. When a key is assigned a different function, its definition within the buffer is altered. A key definition can contain up to 15 characters. If you PEEK into the buffer's memory locations, you may be surprised to find that each key is assigned not 15, but 16 positions. We'll explain why in a moment. In the meantime, knowing the number of positions allotted for each function key makes it

easy to save the buffer's contents—and therefore to preserve the keys' definitions.

Program 2 does this by reading the contents of the buffer into an array. It then assigns new functions to the keys (nonsense definitions for this example). Finally, the program lets you restore the original functions by POKEing the contents of the array back into the soft key buffer. You can use this technique in your own programs to restore the function keys.

If you're still wondering why each key is assigned 16 positions in the buffer when its definition can be only 15 characters long, disabling the keys will provide the answer. If you PEEK at the 16 positions reserved for F1 (originally defined as LIST) and print out the ASCII values, this is what you'll see:

#### LIST0000000000000

When you disable F1, the buffer looks like this:

#### 0 I S T 0 0 0 0 0 0 0 0 0 0 0 0

This seems to indicate that BASIC marks the end of a function key definition with a zero. To prove this, run Program 3. It demonstrates that you can restore the function keys after disabling them by merely saving the first character of each key definition (assuming, of course, that the keys have been disabled by overwriting only the first character of the definition). That's why Program 3 needs to save only 10 bytes instead of the 160 bytes saved by Program 2.

## **Extended Definitions**

Knowing that you can restore the disabled function keys by saving only the first character of each definition may be interesting, but the difference between 10 and 160 bytes probably is of little concern to you. The real power in this knowledge is that you can extend the number of characters available for a function key's definition by altering the sixteenth position in the buffer for that key. This lets you assign a longer definition to a function key (at the expense of the following key, however).

For instance, I prefer to edit programs in SCREEN 0,0,0 and WIDTH 80. Using Program 4, I can set F9 to execute these commands even though they exceed 15 characters. F10 becomes useless, since we haven't increased the size of the soft key buffer—just the length of F9's definition within that buffer.

Program 4 also lets you save the new function key definitions as a file which can be BLOADed from another program. If you try this, don't omit the buffer address (1619) when BLOADing the file, since there's no way to insure that BA-SIC's segment will be the same as when you originally created the file.

#### **Program 1. Buffer Finder for Compatibles**

For error-free program entry, be sure to use "The Automatic Proofreader," Appendix B.

```
PC 100 DEF SEG: SCREEN 0: WIDTH 80: X=0
```

- OH 110 CLS: PRINT "MEMORY LOCATION ":: LOCATE , 20
- 10 120 KEY 1, "LIST": A=ASC("L")
- 16 13Ø IF PEEK(X)=A THEN GOSUB 15Ø ELSE PRINT X;:
  LOCATE 1,2Ø
- J6 14Ø X=X+1:GOTO 13Ø
- ON 150 IF CHR\$(PEEK(X+1))<>"I" THEN RETURN
- PG 160 IF CHR\$(PEEK(X+2))<>"S" THEN RETURN
- CE 170 IF CHR\$(PEEK(X+3))<>"T" THEN RETURN
- JK 180 CLS: PRINT "MEMORY LOCATION "; X
- NH 190 FOR J=1 TO 10:PRINT "F"; J;:FOR K=0 TO 15
- AH 2000 IF PEEK(X+16\*(J-1)+K)>0 THEN PRINT CHR\$(PEEK(X+16\*(J-1)+K)); ELSE 220
- 00 21Ø NEXT K
- PP 220 PRINT: NEXT J
- 66 230 BEEP: INPUT "IS THIS IT ":Q\$
- NO 240 IF Qs="Y" OR Qs="y" THEN END ELSE X=X+1:CL S:GOTO 110

# **Program 2. Restoring Function Definitions**

- HO 90 REM LINES WHICH USE 1619 OFFSET ARE 140 AND 250
- B 100 SCREEN 0: WIDTH 80:CLS:DEF SEG:OPTION BASE
- M 110 KEY ON:DIM K\$(10):FOR X=1 TO 10:K\$(X)=STRI
  NG\$(16,0):NEXT X:' STORAGE AREA FOR FUNCTI
  ON KEYS
- 86 120 REM SAVE FUNCTION KEYS
- HL 13Ø FOR X=1 TO 10:FOR J=Ø TO 15
- EH 140 MID\$(K\$(X),J+1,1)=CHR\$(PEEK(1619+16\*(X-1)+J))
- OP 150 NEXT J, X
- HA 160 REM REDEFINE FUNCTION KEYS WITH LETTERS (T HIS IS ONLY AN EXAMPLE)
- 1£ 170 FOR X=1 TO 10:KEY X,CHR\$(X+64):NEXT X:KEY
  LIST

- EN 180 PRINT "Function keys are redefined":PRINT "Press any key to restore"
- ₩ 19Ø KB\$=INKEY\$: IF KB\$="" THEN 19Ø
- PF 200 REM RESTORE FUNCTION KEYS
- OF 210 FOR X=1 TO 10
- CE 220 KEY X,K\$(X)
- H 230 NEXT X:CLS
- PL 24Ø FOR X=1 TO 1Ø
- NE 25Ø J=ASC(MID\$(K\$(X),16,1)):IF J>Ø THEN POKE 1 619+16\*(X-1)+15,J
- HC 260 NEXT X:CLS
- EL 270 KEY LIST

## **Program 3. Restoring Function Definitions**

- 0L 90 REM LINES WHICH USE 1619 OFFSET ARE 140 AND 220
- EK 100 SCREEN 0: WIDTH 80: CLS: DEF SEG
- % 110 KEY ON:K\$=STRING\$(10,0): STORAGE AREA FOR FUNCTION KEYS
- 86 120 REM SAVE FUNCTION KEYS
- PI 13Ø FOR X=1 TO 1Ø
- N 140 MID\$(K\$, X, 1)=CHR\$(PEEK(1619+16\*(X-1)))
- 朗 15Ø NEXT X
- NJ 160 REM DISABLE FUNCTION KEYS
- HE 170 FOR X=1 TO 10:KEY X,"":NEXT X:KEY LIST
- M 180 PRINT "Function keys are disabled":PRINT " Press any key to restore"
- ₩ 19Ø KB\$=INKEY\$: IF KB\$="" THEN 19Ø
- PF 200 REM RESTORE FUNCTION KEYS
- OF 210 FOR X=1 TO 10
- PO 220 POKE 1619+16\*(X-1), ASC(MID\$(K\$, X, 1))
- M 23Ø NEXT X:CLS
- OF 240 KEY LIST

# **Program 4. Extending Definitions**

- © 9Ø REM LINES WHICH USE THE 1619 OFFSET ARE 18Ø ,29Ø,39Ø,44Ø,47Ø
- IF 100 DEF SEG:STK\$=STRING\$(128,0):SCR\$=STRING\$(3
  7,0):RESTORE 110:FOR X=1 TO 37:READ J:MID\$
  (SCR\$,X,1)=CHR\$(J):NEXT X:SCR!=PEEK(VARPTR
  (SCR\$)+1)+256\*PEEK(VARPTR(SCR\$)+2)
- LG 110 DATA 85,137,229,139,118,6,41,192,138,4,139
- NB 12Ø DATA 1,24Ø,137,196,184,Ø,6,187,Ø,7,185,Ø,2
- FP 130 DATA 186,80,24,85,205,16,92,93,202,2,0
- CE 140 SCREEN 0: WIDTH BO: CLS

```
H! 150 T$="Function Key Definition"
FH 160 LOCATE 2, (40-.5*LEN(T$)):PRINT T$
PP 170 PRINT: PRINT
HE 180 X=1:J=1:K=1619
EF 190 K$=STRING$ (160,0): KN$=STRING$ (160,0): K=K-1
CP 200 L=PEEK (J+K)
PN 210 WHILE L<>0
PH 220 MID$ (K$, J, 1) = CHR$ (L)
OK 23Ø J=J+1:L=PEEK(J+K)
EN 240 WEND
ID 25Ø PRINT "Function Key "; X; ": "; MID$(K$,1,J-1
L! 260 PRINT:PRINT "Enter new definition or press
       ENTER to leave unchanged"
DB 270 LINE INPUT Q$: IF LEN(Q$) >0 THEN GOSUB 300:
      IF ER=1 THEN ER=0:GOTO 250
LF 28Ø IF X+FIX(J/16)>9 THEN GOTO 38Ø
JO 290 X=X+1+FIX(J/16):K=1619+16*(X-1)-1:J=1:CALL
       SCR! (STK$):LOCATE 5,1:GOTO 200
DD 300 INPUT "Do you want a carriage return (Y/N)
       ":01$
SF 310 IF Q1$="Y" OR Q1$="y" THEN Q$=Q$+CHR$(13)
JJ 320 IF LEN(Q$)<16 THEN J=LEN(Q$):KEY X,Q$:RETU
      RN
张 33Ø M=1:N=16*(X-1)+1:IF N+LEN(Q$)>16Ø THEN BEE
       P:PRINT "Too long": ER=1:RETURN
PC 340 MID$ (KN$, N, 1) = MID$ (Q$, M, 1)
# 35Ø M=M+1:N=1+N:IF M<=LEN(Q$) THEN 34Ø
BN 360 IF LEN(Q$)>J THEN J=LEN(Q$)
NL 37Ø RETURN
PE 38Ø FOR X=1 TO 1Ø
NP 39Ø IF ASC(MID$(KN$,16*(X-1)+1,1))>Ø THEN FOR
       J=16*(X-1)+1 TO 16*X:POKE 1619+J-1,ASC(MID
       $(KN$, J, 1)):NEXT J
FA 400 NEXT X:CLS:KEY LIST
JE 410 KB$=INKEY$: IF KB$="" THEN 420 ELSE 410
IF 420 PRINT: INPUT "Do you want to save function
       keys as a BLOADable file (Y/N)":Q$
GJ 430 IF Qs="Y" OR Qs="y" THEN INPUT "Filename";
       F$ ELSE END
M 440 BSAVE F$, 1619, 159: PRINT
EN 450 PRINT "To load your function key file, use
        these commands:"
AA 46Ø PRINT:PRINT
NF 470 PRINT "DEF SEG: BLOAD "; CHR$ (34); F$; CHR$ (34
```

); ", 1619: CLS": END

# Variable Snapshot

Tony Roberts

This programming utility lets you list the current values of all active variables in any BASIC program—an invaluable aid for debugging. It works on any IBM PC with BASICA or PCjr with Cartridge BASIC.

When things go haywire with a BASIC program, my first inclination is to check the variables: PRINT A\$, PRINT SCORE, PRINT UPPERLIMIT, and so on. Comparing what's actually stored in a variable with what you expected often helps to isolate programming problems.

Printing variable values over and over, however, quickly becomes tedious, especially when arrays are involved. "IBM

Variable Snapshot" takes the work out of this process.

After temporarily appending the Variable Snapshot utility to your program, you can activate it with a simple GOTO statement whenever your program stops with an error or you press the Break key. Once activated, Variable Snapshot sifts through memory, first printing out the scalar variables, then the array variables it finds. Within seconds, you can see the values of all the variables your program has used. This kind of analysis has many benefits:

• By frequently checking the variable list, you reduce the possibility of "forgotten" variables.

You can quickly spot typographical errors in variable names.
 If the list contains both FILENAME\$ and FILENAM\$, you'll

realize something is wrong.

 By checking variable types as well as names, you'll notice if the list contains both TOTAL% (an integer variable) and TO-TAL! (a single-precision variable)—another common source of errors.

How to Take Snapshots

Type in Program 1 and save it to disk in ASCII format. If you type it in with the "Automatic Proofreader," found in Appendix B, the program is saved in ASCII format automatically. Otherwise, use the command SAVE "SNAPSHOT.ASC", A.

Program 2 lets you test Variable Snapshot to verify that it's working properly before using it with your own programs. To run a test, type in Program 2 and save it to disk in ASCII format. Append Snapshot to it with the command MERGE "SNAPSHOT.ASC". Now type RUN. The test program initializes several variables, then stops. When you type GOTO 1000 (the starting line number of Variable Snapshot), the name and value of each variable is printed on the screen. You can press Ctrl-Num Lock on the PC or Fn-Q on the PCjr to pause the display, or stop it at any time by pressing Ctrl-Break on the PC or Fn-Break on the PCjr.

If the variable values are not what you expected, recheck your typing, paying close attention to the type declaration symbols (%, \$, !, #) attached to the variables. If even one of these symbols is incorrect or missing, you'll have problems.

The test program initializes integer, string, single-precision, and double-precision variables as well as a full set of array variables. If everything prints out as expected, you can be pretty sure that Variable Snapshot is working well.

Variable	Description
Q%,QQ%,QQQ%	Loop counters
QTYPE%	Variable type
QLENLEFT%	Number of characters left in variable name
QDIMS%	Number of dimensions in array
QARRAYON%	Flag indicating if array boundary passed
QSTRLEN%	Length of string variable
QBASE%	Status of OPTION BASE command
Q\$	For single- and double-precision conversions
QCHAR\$	Builds active filename
QFILE\$	Active filename
QNAME\$	Name of variable being processed
QVAR!	Memory pointer to current variable
QARRAY!	Start of array space
QFREE!	Start of free space
QASIZE!	Size of current array
QVALUE!	Temporary storage for integer values
QSTRPTR!	Points to location of actual string
QPTR!	Points to start of next element in array
QDIMSIZE%( )	Size of array dimensions

Friendly Filename and Quick Start

When Snapshot begins its work, the first thing it prints is the active disk filename, which the IBM stores in the 11 memory locations beginning at 4F1h (1265 decimal). This has nothing at all to do with variables, but simply provides an answer to the question What did I call this program the last time I saved it?

If you want to start Snapshot quickly, you can omit the entire array processing section (lines 1590–2220) and change line 1280 to read:

#### 1280 IF QARRAYON% THEN END

This abbreviated version of Snapshot lists only simple variables, but you can go back later and add the lines to handle the array variables. The REMs in the program listing are not referenced by other lines, so you can safely omit them when typing the program.

After you have Snapshot working, edit line 1000 to suit

your preferences for screen color, width, and so on.

You may want to renumber Snapshot so that its line numbers won't interfere with those of your own programs. (Low line numbers were used in the listing to make entering the program easier.) Load the program into memory and use the command RENUM xxxxx, where xxxxx is Snapshot's new starting line number. Then save the program back to disk, again using the ASCII option so that Snapshot can be merged with other programs.

The version I use begins at line 60000, and I've programmed a function key to execute the command GOTO 60000 (see the article "Advanced Function Key Techniques" for more information on programming these keys). Whenever a program halts, I simply press Fn-6 to see the value of every

variable.

**Array Bases** 

IBM BASIC includes the OPTION BASE statement for defining the lowest-numbered element in an array. If a program contains the statement OPTION BASE 0, or if no OPTION BASE statement is included, all arrays start with a zero element. An OPTION BASE 1 statement means that arrays begin with element 1.

Variable Snapshot must know which OPTION BASE is in effect to display array values properly. Memory location 45Ch

(1116 decimal) provides this information. PEEKing that address yields either a zero or one, indicating which base is selected.

The adjacent memory location, 45Dh (1117 decimal), is related but a little more specific. If no OPTION BASE command has been issued, 45Dh contains a zero; if OPTION BASE 0 has been executed, 45Dh contains a one; and if OPTION BASE 1 has been executed, the location contains a two.

Try changing line 20 in Program 2 to read OPTION BASE 0 and observe the effect when running Variable Snapshot.

Although IBM BASIC allows arrays of up to 255 dimensions, few programs make use of more than one or two. For this reason, Variable Snapshot does not include provisions for arrays with more than two dimensions. Additional loops can be added to handle more complex arrays, if necessary.

#### A Few Cautions

To be truthful, Snapshot does not list *every* variable—it ignores those that begin with the letter Q. The Snapshot routine itself, you'll notice, uses only variables beginning with the letter Q. That keeps Snapshot's own variables from being printed along with those of your program.

If you're inclined to tinker with this routine, you must be careful about introducing new variables. Lines 1020–1040 initialize every variable used by the routine, effectively reserving

space for them in the variable table.

Lines 1120–1140 determine the boundaries of the variable table, reference points the program cannot do without. If a new variable is added to the program after the boundary measurements are taken, confusion results; the boundaries move and Snapshot loses its way.

Although Snapshot works with most programs, there can be complications. If you've written your program to make use of all available memory, there won't be room in the variable table for Snapshot's own variables. You'll need to leave Snapshot about 300 bytes of workspace.

**How Snapshot Works** 

As mentioned above, Snapshot reads the boundaries of the scalar variable area, the array variable area, and the free space area, then works its way through the variable areas, byte by

byte, deciphering the information stored there. Once it reaches free space, its work is finished.

The IBM stores scalar variables like this:

```
Byte 1 = type (2 = integer, 3 = string, 4 = single precision, 8 = double precision)

Byte 2 = first character of variable name

Byte 3 = second character of variable name

Byte 4 = number of characters remaining in variable name

Byte 5

= rest of variable name (high bit set)
```

Following the last character of the variable name is the value of the variable.

- An integer variable is stored in two bytes in the standard low byte/high byte format. The high bit of the second byte indicates the sign of the integer. If it's set, the integer is a negative number.
- String variable pointers are stored in three bytes. The first is the number of bytes in the string, and the second and third point to the address (either in the string pool or in the BASIC program area) where the string is stored.
- Single-precision variable values are stored in four bytes. The values of these bytes can be concatenated into a string, then converted into a single-precision number using the CVS function.
- Double-precision variables occupy eight bytes, which can be concatenated and converted as above using the CVD function.
- Array variables are stored similarly, but there's some additional information between the end of the variable name and the actual beginning of the variable values.

Following the variable name are two bytes that indicate the total size of the array. The next byte holds the number of dimensions. That's followed by two bytes describing the number of elements in the last dimension. Then two bytes describe the number of elements in the next to last dimension, and so on, until each dimension in the array has been defined.

Finally, the values of the array variables follow, and are stored in the same manner as values for scalar variables.

#### Program 1. IBM Variable Snapshot

For error-free program entry, be sure to use "The Automatic Proofreader," Appendix B.

- Ck 1000 DEF SEG: SCREEN 0,0:WIDTH 80:COLOR 7,0:
- B 1010 REM initialize variables
- NE 1020 Q%=0:QQ%=0:QQQ%=0:QLENLEFT%=0:QTYPE%=0:QD IMS%=0:QARRAYON%=0:QSTMLEN%=0:QBASE%=0:QD IMSIZE%(1)=0:QDIMSIZE%(2)=0
- LN 1030 Qs="":QCHAR\$="":QFILE\$="":QNAME\$=""
- \*\*\* 1040 QVAR!=0:QARRAY!=0:QFREE!=0:QASIZE!=0:QVAL UE!=0:QSTRPTR!=0:QPTR!=0
- 66 1050 REM Get active filename
- 0L 1060 FOR Q%=Ø T● 1Ø
- 84 1070 QCHAR\$=CHR\$(PEEK(&H4F1+Q%))
- PA 1080 IF ASC(QCHAR\$)>96 AND ASC(QCHAR\$)<123 THE N QFILE\$=QFILE\$+CHR\$(ASC(QCHAR\$)-32) ELSE QFILE\$=QFILE\$+QCHAR\$
- AH 1090 NEXT
- II 1100 PRINT:PRINT "Active disk filename is: ";
  MID\$(QFILE\$,1,8);".";MID\$(QFILE\$,9):PRINT
- CM 1110 REM get addresses of scalar variables, ar ray variables, and free space
- FH 1120 QVAR!=PEEK (&H358) +PEEK (&H359) \*256
- HI 1130 QARRAY!=PEEK(&H35A)+PEEK(&H35B) \*256
- HD 1140 QFREE!=PEEK(&H35C)+PEEK(&H35D) \*256
- HF 1150 QBASE%=PEEK (&H45C)
- BN 1160 REM Start of variable processing
- CH 1170 QTYPE%=PEEK (QVAR!)
- J0 1180 IF (QTYPE%<2 OR QTYPE%>4) AND QTYPE%<>8 THEN END
- LG 1190 QLENLEFT%=PEEK(QVAR!+3)
- PH 1200 REM get variable name
- QN 1210 QNAMES=""
- L 1220 IF PEEK(QVAR!+1)>127 OR (PEEK(QVAR!+1)=81 AND QARRAYON%=0) THEN 2240
- EC 1230 FOR Q%=1 TO QLENLEFT%
- #J 1240 QNAME\$=QNAME\$+CHR\$(PEEK(QVAR!+3+Q%) AND 1 27)
- ₽ 125Ø NEXT
- NE 1260 QNAMES=CHR\$ (PEEK (QVAR!+1))+CHR\$ (PEEK (QVAR!+2))+QNAME\$
- II 1270 REM branch to appropriate routine depending on variable type
- EG 128Ø IF QARRAYON% THEN 16ØØ
- CF 1290 ON QTYPE%-1 GOTO 1320,1370,1460
- NP 1300 GDTO 1530
- FM 1310 REM inteters
- EC 132Ø QVALUE!=PEEK(QVAR!+QLENLEFT%+4)+PEEK(QVAR
  !+QLENLEFT%+5)\*256
- 09 1330 IF QVALUE!>32768! THEN QVALUE!=QVALUE!-45
  536!

```
PH 1340 PRINT QNAME$; "%"; , "= "; QVALUE!
№ 135Ø GOTO 224Ø
ff 1360 REM strings
PH 1370 PRINT QNAMEs; "$", "= "; CHR$ (34);
BN 1380 QSTRLEN%=PEEK(QVAR!+QLENLEFT%+4)
MC 1390 QSTRPTR!=PEEK(QVAR!+QLENLEFT%+5)+PEEK(QVA
       R!+QLENLEFT%+6) *256
SJ 1400 FOR Q%=0 TO QSTRLEN%-1
PM 1410 PRINT CHR$ (PEEK (QSTRPTR!+Q%)):
QK 142Ø NEXT
61 143Ø PRINT CHR$ (34)
NB 144Ø GOTO 224Ø
N 1450 REM single precision
î 146Ø Q$=""
FG 147Ø PRINT QNAMES; "!", "= ";
CM 1480 FOR Q%=0 TO 3: Q$=Q$+CHR$(PEEK(QVAR!+QLEN
       LEFT%+4+Q%))
BP 149Ø NEXT
BB 1500 PRINT CVS(Q$)
NK 151Ø GOTO 224Ø
KB 1520 REM double precision
Œ 153Ø Q$=""
HH 1540 PRINT QNAME$; "#", "= ";
GN 1550 FOR Q%=0 TO 7: Q$=Q$+CHR$(PEEK(QVAR!+QLEN
       LEFT%+4+Q%))
B! 1560 NEXT
FE 1570 PRINT CVD(Q$)
OP 158Ø GOTO 224Ø
AC 1590 REM array routines
M 1600 QASIZE!=PEEK(QVAR!+4+QLENLEFT%)+PEEK(QVAR
       !+5+QLENLEFT%) *256
JI 161Ø IF ASC (QNAME$) = B1 THEN 224₽
BA 1620 QDIMS%=PEEK (QVAR!+6+QLENLEFT%)
18 163Ø IF QDIMS%>2 THEN 224Ø
QL 1640 QPTR!=QVAR!+7+QLENLEFT%
NA 1650 FOR Q%=QDIMS% TO 1 STEP -1
LN 1660 ODIMSIZE%(Q%)=PEEK(QPTR!)+PEEK(QPTR!+1)*2
       56
NE 167Ø QPTR!=QPTR!+2
BA 168Ø NEXT
J6 1690 ON QTYPE%-1 GOTO 1720,1830,1980
KI 1700 GOTO 2110
A 1710 REM integer arrays
AB 172Ø PRINT
BL 1730 IF QDIMS%=2 THEN FOR QQQ%=QBASE% TO QDIMS
       IZE%(2) + (QBASE%=0)
61 1740 FOR Q%=QBASE% TO QDIMSIZE%(1)+(QBASE%=0)
ON 1750 QVALUE!=PEEK(QPTR!)+PEEK(QPTR!+1)*256
EA 1760 IF QVALUE!>32768! THEN QVALUE!=QVALUE!-65
```

536!

```
IF 1770 IF QDIMS%=1 THEN PRINT QNAMES; "%("; MID$(S
       TR$(Q%),2);")","= ";QVALUE! ELSE PRINT QN
       AME$: "%(":MID$(STR$(Q%),2);",";MID$(STR$(
       QQQ%).2):")"."= ":QVALUE!
N 1780 QPTR!=QPTR!+2
JN 1790 NEXT Q%
EH 1800 IF QDIMS%=2 THEN NEXT QQQ%
NA 1810 GOTO 2240
MA 1820 REM string arrays
AG 1830 PRINT
BA 1840 IF QDIMSX=2 THEN FOR QQQX=QBASEX TO QDIMS
       IZE%(2)+(QBASE%=Ø)
GN 1850 FOR Q%=QBASE% TO QDIMSIZE%(1)+(QBASE%=0)
IE 1860 QSTRLEN%=PEEK (QPTR!)
KA 1870 QSTRPTR!=PEEK(QPTR!+1)+PEEK(QPTR!+2) *256
0 1880 IF QDIMSX=1 THEN PRINT QNAMES; "$(":MID$(S
       TR$(Q%),2);")","= ";CHR$(34); ELSE PRINT
       QNAME$; "$("; MID$(STR$(Q%), 2); ", "; MID$(STR
       $(QQQ%),2);")","= ";CHR$(34);
CI 1890 FOR QQ%=0 TO QSTRLEN%-1
ME 1900 PRINT CHR$ (PEEK (QSTRPTR!+QQ%)):
HJ 1910 NEXT 00%
SP 1920 PRINT CHR$ (34)
9A 193Ø QPTR!=QPTR!+3
18 194Ø NEXT Q%
GI 1950 IF QDIMSX=2 THEN NEXT QQQX
08 196Ø GOTO 224€
EL 1970 REM single precision arrays
BH 1980 PRINT
CB 1990 IF QDIMS%=2 THEN FOR QQQ%=QBASE% TO QDIMS
       IZEX(2) + (QBASEX = \emptyset)
EP 2000 FOR Q%=QBASE% TO QDIMSIZE%(1)+(QBASE%=0)
NF 2Ø1Ø Q$=""
IH 202# FOR QQ%=0 TO 3
08 2030 Q$=Q$+CHR$ (PEEK (QPTR'+QQ%))
68 2040 NEXT QQ%
AH 2050 IF QDIMS%=1 THEN PRINT QNAMES; "!("; MID$(S
       TR$(Q%),2);")","= ";CVS(Q$) ELSE PRINT QN
       AME$; "! ("; MID$ (STR$ (Q%), 2); ", "MID$ (STR$ (Q
       QQ%),2);")","= ";CVS(Q$)
PK 2060 QPTR!=QPTR!+4
IJ 2070 NEXT Q%
FA 2080 IF QDIMSX=2 THEN NEXT QQQX
OJ 2090 GOTO 2240
NG 2100 REM double precision arrays
PD 2110 PRINT
WH 2120 IF QDIMSX=2 THEN FOR QQQX=QBASEX TO QDIMS
       IZE%(2)+(QBASE%=Ø)
FK 2130 FOR Q%=QBASE% TO QDIMSIZE%(1)+(QBASE%=0)
0A 214Ø Q$=""
```

00 215Ø FOR QQ%=Ø TO 7 DM 2160 Q\$=Q\$+CHR\$(PEEK(QPTR!+QQ%)) HM 2170 NEXT QQ% W 2180 IF QDIMS%=1 THEN PRINT QNAMES; "#("; MID\$(S TR\$(Q%),2);")","= ";CVD(Q\$) ELSE PRINT QN AMEs: "#(":MIDs(STRs(Q%),2);",";MIDs(STRs( QQQ%),2);")","= ";CVD(Q\$) EN 2190 QPTR!=QPTR!+8 HI 2200 NEXT Q% EP 2210 IF QDIMS%=2 THEN NEXT QQQ% NI 222Ø GOTO 224Ø GH 2230 REM Get address of next variable MM 224Ø IF QARRAYON%<>1 THEN QVAR!=QVAR!+QLENLEFT %+QTYPE%+4 ELSE QVAR!=QVAR!+QASIZE!+QLENL EFT%+6 HH 2250 IF QVAR!=>QARRAY! THEN QARRAYON%=1 NG 2260 IF QVAR!=>QFREE! THEN END OH 2270 GOTO 1170

#### Program 2. Snapshot Demo

# IBM BASIC Tips and Techniques

C. Regena

C. Regena, columnist for COMPUTE! magazine and author of numerous BASIC programming books, offers a collection of BASIC tidbits and tricks for IBM users.

# **Programming Hints and Tips**

- All BASIC lines must be numbered. Line numbers may be from 0 to 65529.
- The command AUTO makes the computer automatically type the line numbers for you as you're programming. AUTO starts the automatic numbering with the current line number and increments by tens. Press Fn-Break to disable automatic numbering. Do not edit while in AUTO mode.
- RENUM renumbers or resequences your program. All line numbers referenced in other statements are automatically renumbered.
- LIST . (that's a period) lists the last line you typed in, edited, or LISTed.
- Use REM to indicate a remark statement. Abbreviate this command by typing the single quotation mark or apostrophe:

## 10'TITLE

You can separate commands on a line with the colon, but if the last statement is a remark, the colon and REM may be replaced by the apostrophe:

70 SC=SC+1 ' INCREMENT SCORE

# Variable Names

- Variable names may be up to 40 characters long. Actually, they may be longer, but only the first 40 characters will be recognized. The variable name may contain embedded reserved BASIC words without any problems.
- Fn-Pause stops a program listing or freezes a running program. Press any key to continue.

• Fn-Break also stops a listing or a program, but it prints a message. To continue, type CONT and press Enter.

• When you use a function key, press Fn first, then the second key. However, when you use Alt or Ctrl, you must hold it

down as you press the second key.

• The Ctrl-Alt-Del key combination (the Control key at the left of the keyboard, the Alt key, and the Delete key) resets the system. The result is just as if you had turned the computer off, then on again.

 Press Ctrl-Alt and either the left-arrow or right-arrow key to move the screen image left or right on the monitor or television screen. With some television sets you need to center

your screen to read all the columns.

• For inequality, you may use either <> or ><.

# Arrays

• An array may have 235 dimensions. The maximum number of elements per dimension is 32767.

• If you use a subscripted number without a previous DIMension statement, the computer assumes DIM X(10), or the 11 elements from 0 to 10. A regular variable may have the same name as an array variable without causing any problems. For example, you can use both A\$ and A\$(3,5) in the same program. They're treated as separate variable names.

• BASIC filenames, such as names of programs you save, may be up to eight characters long. Disk filenames may be eight characters long plus a period and a three-character extension.

• To use the disk drive and program in BASIC on the PCjr, you must have Cartridge BASIC. Before you can save a program, the disk must be *initialized*, or *formatted*. Turn on the computer with the DOS disk in the drive. The FORMAT command is part of DOS (Disk Operating System).

 FORMAT has several options. FORMAT /S formats the disk and transfers the DOS commands to your program disk so you don't need to insert the DOS disk when returning to

DOS.

• The /V option when formatting allows you to put a title on a disk (Volume label). After the computer formats the disk, it prompts you for a title, which may be up to 11 characters long. Type FORMAT /S/V to use both the system option and the volume label option.

 To save a program in protected form, use P after the filename. This protection keeps someone from listing, editing, or saving the program. This is available in Cassette BASIC or Cartridge BASIC, on cassette or disk. An example is SAVE "MYGAME",P.

#### **Automatic Load and Run**

What follows is a procedure to load and run a program automatically after you turn on the computer:

1. Type in or load your BASIC program. For an example, try this program.

10 REM PRINTS DIRECTORY

20 CLS

30 FILES

**40 NEW** 

50 END

2. Save your program normally on the disk.

SAVE "DISK"

3. Type SYSTEM and press Enter to return to DOS.

4. Type COPY CON: AUTOEXEC.BAT and press Enter.

5. Type BASIC DISK and press Enter—or use the name of the program you want to run automatically.

6. Press Fn-6, then press Enter. You'll see a *Z* on the screen. The drive will whirl momentarily.

Now, when you insert the disk at the beginning of a session, the program you named automatically loads and runs.

The sample program above prints a directory of the files on the disk. Line 40 erases the program so you can enter or load another program.

# Using the Function Keys

 Remember to use the function keys to save typing. To load a program, press Fn-3, then type the name of the program.
 You don't need to type the last quotation mark, and the name may be in uppercase or lowercase letters.

• Fn-1 is LIST, and you may specify lines or list the whole program. Fn-2 is RUN and includes Enter. Fn-3 is LOAD", and Fn-4 is SAVE". Just fill in the name of the program. Fn-5 is CONT and is used to continue running a program after you've pressed Fn-Break.

- Fn-6 is ,"LPT1:" and Enter. You type in the beginning of the command for what you want on the printer. Fn-7 is TRON, and Fn-8 is TROFF, and you don't need to press Enter. These are TRACE commands for debugging programs. Fn-9 is the word KEY; add whatever you need to complete the command (such as KEY OFF). Fn-10 is SCREEN 0,0,0 and Enter, which is very handy if you're working in a graphics mode and want to clear the screen and return to the normal text screen.
- If you've forgotten what files or programs are on your disk, use the DOS command DIR (for directory). The files or program names will be listed, along with the size of each file and the date and time you saved it. If you're in BASIC and need to get to DOS, type SYSTEM and press Enter, then type DIR for directory. If you don't want to lose your BASIC program by typing SYSTEM, type the command FILES to see the disk directory. Only the filenames are shown.

#### PRINT USING

Quite a few commands are available in Cassette BASIC which are not discussed in the *Hands-On BASIC* book that comes with the PCjr. The *BASIC Reference Guide* that comes with Cartridge BASIC is a necessity if you like to program.

One useful command is PRINT USING, which can help format your output. You can use commas, semicolons, and the TAB function in your regular PRINT statements to line up columns and make the screen look nice, but often PRINT USING is easier.

One of PRINT USING's handiest features is right-justification of a column of numbers (lining up the decimal places). A regular PRINT statement will start printing the number in the next print position. The numbers will line up fine—as long as their lengths are the same (for example, all two-digit numbers between 15 and 70). PRINT USING will right justify the numbers so that the ones column lines up, the tens column lines up, and so forth. The basic command with numbers is

#### PRINT USING "###";N

where *N* is a variable name for a number. The # signs each represent a digit. This format allows for a three-digit number.

You may specify a certain number of decimal places—and

the computer will round (that's round, not truncate) to that number of places. PRINT USING "##.###" will print numbers under 100 and allow up to three decimal places. If the number rounds to 100 or more, a percent sign will be printed before the number, but the number will still be printed correctly. If the number is less than one, a zero will be printed to the left of the decimal.

If you're printing large numbers, you may want to use commas in your numbers for thousands or millions or billions. As you know, the computer normally prints the number with no commas. You could use string functions and figure out the length of the number (number of digits), then combine parts of strings to get the number with commas in the right places. But PRINT USING can automatically place commas in large numbers. Use a comma just before the decimal point, and a comma is printed to the left of every third digit to the left of the decimal point. PRINT USING "####,.##" places commas for the thousands.

#### **Dollars and Cents**

- If you use your computer to print reports involving money, PRINT USING "\$ ###.##"; PRICE prints the number rounded to two decimal places and puts a dollar sign and a space before the number. By the way, specifying the two places after the decimal point makes sure that zeros will be printed, if necessary, to fill in the places. A price that would ordinarily turn out to be 256.2 will be printed \$ 256.20 in this format. Two dollar signs just before the number symbols print the dollar sign immediately before the number.
- You may combine a message with the number, such as PRINT USING "THE ANSWER IS ##.##"; A. You can also put a plus sign (+) before or after the number symbols to print the sign of the number before or after the number.
- Two asterisks before the number symbols cause leading spaces to be filled in with asterisks. There are other options available, too, so you might take a few minutes and experiment with them to become familiar with this handy command.

# Screen Swapping

Paul W. Carlson

If you've ever needed to store a graphics screen temporarily for later recall in a program, or load screens from disk and flash them on the monitor whenever you want, this article shows you how. The programs work on any IBM PC with color/graphics adapter and BASICA or the Enhanced Model PCjr with Cartridge BASIC.

You can achieve many interesting effects, including animation, by rapidly switching between several graphics screens stored in memory. Unfortunately, this capability isn't a standard feature on the IBM PC. With help from two very short machine language subroutines, however, you can write programs that swap screens almost instantly. The subroutines copy the video bitmap to or from an array in about five thousandths of a second, much too fast for the eye to see. In fact, this is even faster than the video monitor can display a frame, so the effect is truly instantaneous.

Type in Program 1. It creates two files, SCRNARRY.BAS and ARRYSCRN.BAS, which contain the two machine language subroutines. The first routine copies the video bitmap to an array and the second copies the contents of an array to the video bitmap. The routines achieve their speed by treating the bitmap as a continuous string of 16,192 bytes.

To see how to use these routines in your own programs, enter Program 2 and save it on the same disk with SCRNARRY.BAS and ARRYSCRN.BAS. Before running Program 2, make sure the disk is in the active drive; it accesses the two routines as it runs. After typing RUN, don't press any keys until you want to halt the program.

You should see three multicolored spirals drawn on the screen. The first two disappear as soon as they're completed, and the third seems to rotate. The rotation, of course, is an illusion.

Here's what happens: In the split second between the time the first two spirals are completed and then erased, each screen is copied into an array by SCRNARRY.BAS. The third spiral is also copied into an array. Finally, the contents of all three arrays are repeatedly copied to the screen by ARRYSCRN.BAS to get the rotating effect. Actually, the program requires a time-delay loop to keep the screen flipping from happening too fast.

Program 2 works like this:

Line(s)	Description
20–30	Load the machine language subroutines into the STOA and ATOS arrays.
40-140	Draw and paint three spirals, each one with the colors shifted.
150	GETSCRN is the entry point for the subroutine that copies the screen to an array. <i>Important:</i> No new simple variables can be assigned from the point GETSCRN is computed to the point it is used in a CALL statement. Assigning simple variables causes array addresses to move.
160-200	Copy the screen to array SCRN1, SCRN2, or SCRN3 after each spiral is complete.
210	PUTSCRN is the entry point for the subroutine that copies an array to the screen. The same note for line 150 applies here also.
220–250	Repeatedly copies the arrays SCRN1, SCRN2, and SCRN3 to the screen until a key is pressed.

# **Computerized Slide Show**

You can load a graphics screen from disk directly into an array the same way Program 2 loads the machine language into arrays. Why would you want to do this? Suppose you had saved graphics screens from three different programs on disk using statements such as

# DEF SEG=&HB800:BSAVE"filename",0,16192

with filenames of PIC1, PIC2, and PIC3. You could then use Program 3 to display a "slide show" of your creations.

This interesting program displays one screen while loading another. Pressing the space bar (after giving the next screen time to load) displays the next picture. The program could be extended to accommodate any number of screens, even prompting you to change disks if necessary. It needs only one array to store the screens, no matter how many you want to display, since it stores only one screen at any moment.

Notice that the statement LA=0 in line 10 of Program 3 prevents the address of the ATOS array from changing after it is assigned a value for PUTSCRN in line 30. (See the note for line 150 in the breakdown of Program 2.)

Programs 4 and 5 show the source code for the SCRNARRY and ARRYSCRN subroutines. They aren't required for use with Programs 1–3; they're listed so machine language programmers can observe the techniques involved. An assembler is required to enter these listings.

#### **Program 1. Screen Swapping Routines**

For error-free program entry, be sure to use "The Automatic Proofreader," Appendix B.

```
SH 10 DIM M(7), J(6):DEF SEG
FK 20 FOR N=0 TO 26:READ B
NI 30 POKE VARPTR(M(0))+N,B:NEXT
90 40 BSAVE"SCRNARRY", VARPTR(M(0)), 27
8N 50 FOR N=0 TO 22:READ B
KL 60 POKE VARPTR(J(0))+N,B:NEXT
8I 70 BSAVE"ARRYSCRN", VARPTR(J(0)), 23:END
KE 80 DATA 6,30,7,30,139,236,184,0
PO 90 DATA 184,142,216,185,160,31,51,246
GH 100 DATA 139,126,8,252,243,165,31,7
KN 110 DATA 202,2,0,6,139,236,184,0
KF 120 DATA 184,142,192,185,160,31,51,255
DE 130 DATA 139,118,6,252,243,165,7,202
8F 140 DATA 2,0
```

# Program 2. Spiral Demo

```
MK 10 DIM SCRN1(4048), SCRN2(4048), SCRN3(4048), STO
     A(7), ATOS(6)
$1 20 DEF SEG: BLOAD"SCRNARRY", VARPTR (STOA (0))
NK 3Ø BLOAD"ARRYSCRN", VARPTR(ATOS(Ø))
HJ 40 KEY OFF:SCREEN 1:COLOR 0,0
N 50 FOR C=1 TO 3:W=C:CLS
NM 60 TP=6.283185:F=80/TP:DA=TP/9:DB=TF/20:A=0
9M 7Ø FOR I=1 TO 9:B=Ø:A=A+DA:PSET(160,100)
6J 8Ø FOR J=1 TO 2Ø:B=B+DB:R=F*B
LN 90 X=160+1.2*R*SIN(A+B):Y=100+R*COS(A+B)
10 100 LINE -(X,Y),3:NEXT J,I
JK 110 CIRCLE (160, 100), 96, 3: A=DA/2
PM 120 FOR I=1 TO 9: A=A+DA
FL 13Ø X=16Ø+1.18*R*SIN(A):Y=1ØØ+.95*R*COS(A)
FA 140 C=C MOD 3+1:PAINT(X,Y),C,3:NEXT I
FN 150 GETSCRN=VARPTR(STOA(0))
60 160 ON W GOTO 170,180,190
LE 170 CALL GETSCRN(SCRN1(0)):GOTO 200
```

```
NK 180 CALL GETSCRN(SCRN2(0)):GOTO 200
EC 190 CALL GETSCRN(SCRN3(0))
JM 200 NEXT C
BJ 210 PUTSCRN=VARPTR(ATOS(0))
NB 220 CALL PUTSCRN(SCRN1(0)):FOR J=0 TO 100:NEXT
PJ 230 CALL PUTSCRN(SCRN2(0)):FOR J=0 TO 100:NEXT
NB 240 CALL PUTSCRN(SCRN3(0)):FOR J=0 TO 100:NEXT
ME 250 IF INKEY$="" THEN 220
MO 260 CLS:SCREEN 0:WIDTH 80:KEY ON:END
```

## Program 3. Slide Show Demo

```
FE 10 DIM SCRN(4043),ATOS(6):LA=0

HP 20 DEF SEG:BLOAD"ARRYSCRN",VARPTR(ATOS(0))

10 30 PUTSCRN=VARPTR(ATOS(0)):LA=VARPTR(SCRN(0))

NA 40 BLOAD"PIC1",LA

FD 50 KEY OFF:CLS:SCREEN 1:COLOR 0,1

AB 60 CALL PUTSCRN(SCRN(0)):BLOAD"PIC2",LA

68 70 IF INKEY$<>" "THEN 70

C6 80 CALL PUTSCRN(SCRN(0)):BLOAD"PIC3",LA

JD 90 IF INKEY$<>" "THEN 90

10 CALL PUTSCRN(SCRN(0))

L1 10 IF INKEY$<>" "THEN 110

LF 120 CLS:SCREEN 0:WIDTH 80:KEY ON:END
```

# Program 4. SCRNARRY Source Code

Note: This source code is provided for information only. It is not required for Programs 1-3. An assembler is required to enter this listing.

```
This subroutine copies 16192 bytes from the video display
   into a BASIC array.
CSEG
        SEGMENT
STOA
        PROC
                 FAR
        ASSUME CS: CSEG
                                  :Save extra segment
        PUSH
                 ES
        PHSH
                 D S
                                  ;Set the extra segment
        POP
                 FS
                                        equal to the data segment
                                  .
        PUSH
                 DS
                                  ; Save the data segment
        MOV
                 BP, SP
                                  ; Make BP point to the stack
        MOV
                 AX,0B800H
                                  ; Set data segment to beginning
        MOV
                 DS,AX
                                      of video RAM.
        MOV
                                  ; Initialize move counter
                 CX,8096
                                  :Initialize source index
        XOR
                 SI,SI
        MOV
                 D1,8[BP]
                                  ; Init. dest. index to array offset
        CLD
                                  :Set direction flag
REP
        MOVSW
                                  ; Move the display to the array
        POP
                 DS
                                  Restore the data segment
        POP
                 ES
                                  ;Restore the extra segment
        RFT
                 2
                                  ;Clean up the stack
STOA
        ENDP
CSEG
        ENDS
        FND
```

#### Program 5. ARRYSCRN Source Code

Note: This source code is provided for information only. It is not required for Programs 1–3. An assembler is required to enter this listing.

```
This subroutine copies 16192 bytes from a BASIC array
   to the video display.
CSEG
         SEGMENT
ATOS
        PROC
                 FAR
        ASSUME CS:CSEG
         PUSH
                 ES
                                   ; Save extra segment
        MOV
                 BP, SP
                                   :Make BP point to stack
                                   ;Set extra segment to beginning
         VOM
                 AX.0B800H
                                       of video RAM.
        MOV
                 ES.AX
                                   ;Initialize move counter
        MOV
                 CX,8096
         XOR
                 DI, DI
                                   ; Initialize destination index
        MOV
                 S1,6[BP]
                                   : Init, source index to array offset
         CLD
                                   Set direction flag
                                   Move the array to the screen Restore extra segment
REP
        MOVSW
         POP
                 ES
         RET
                                   Clean up stack
                 2
ATOS
         ENDP
CSEG
         ENDS
         END
```

# **Filecopy**

John and Jeff Klein

Here's a fast and easy way to back up multiple files on your disks for safekeeping. It works on any IBM PC, PCjr, or compatible with at least 64K RAM and one or two floppy disk drives.

What is rule number one when you use a computer? Always make backup copies of all important files.

But despite one of the most powerful disk operating systems in personal computing, that rule isn't always easy to follow on the IBM. DOS's DISKCOPY utility indiscriminately copies the entire disk, while the COPY command backs up only individual files. Neither allows you to copy groups of specific files from disk to disk or directory to directory very easily. Even if you have two drives, it's time-consuming to combine files from several disks onto a single backup disk, or to back up a group of updated files. As a result, many of us don't make backups as often as we should.

"IBM Filecopy" offers a solution. It's a utility program that works on any IBM PC, PCjr, or true compatible with one or two disk drives. Filecopy lets you back up disks, directory by directory, or selectively back up individual files. These files can be of any type: BASIC, binary, command, and so on. The files can be copied to any subdirectory on any other disk or the same disk.

Using Filecopy

Filecopy is a BASIC program which creates a temporary DOS batch file to copy the specified files to the backup destination. When run, the program first asks you to insert the *source disk*. This is the disk which contains the files you want to back up. Then the program asks for the source directory of the source disk. If you're not copying from a subdirectory, just type N at this prompt. In either case, Filecopy reads the filenames from the source directory and stores them in an array for later use.

Next, the program asks you for the *target path*, the destination for the backups. Type B: for drive B or A: for drive A (be sure to type the colon after the drive designator—B: instead of B). Then type  $\setminus$  *directory name* if you're copying the

files to a subdirectory on the destination disk. You don't have to specify a directory if you're copying the files to the root (default) directory. If you're using a single-drive system, type *B*: for the target path as if you really have two drives. Never specify the same drive and directory as the source drive and directory, because the program won't copy files onto themselves. (See Table 1 for sample copy procedures.)

# Table 1. Using IBM Filecopy

Type of copy wanted:

Same drive, directory "TEST"

Drive B, same directory

Drive B, directory "TEST"

Target path to enter:

TEST (DOS 2.1 only)

B:

B: \TEST

Filecopy then displays each filename from the source directory and asks if you want a copy. Simply type Y for each file you want copied or N for those you don't want copied.

Note: When Filecopy encounters a subdirectory name on your source disk, it's fooled into thinking the subdirectory is a regular file. Since it can't copy subdirectory names, you must answer N when the program asks whether you want to back up the subdirectory.

After Filecopy queries you on all of the filenames, it asks for confirmation: *Is this all okay?* If you accidentally typed a wrong Y or N at a previous prompt, type N here to get another chance. When you confirm your choices by typing Y, Filecopy stores the names of the files you want copied in a temporary batch file on the source disk. (If the source disk is write-protected, an error message appears and the program halts.) Then it returns you to DOS.

The next step is to type FILECOPY at the DOS prompt. This commands the batch file to copy each of the files you selected from the source disk to the target disk and directory. If you're using a single-drive system, DOS asks you to swap disks as it copies each file.

After the backup is complete, the temporary batch file erases itself from the source disk. This brings up an unavoidable DOS error message, *Insert disk with batch file and press any key when ready*. Just press Ctrl-Break on the PC or Fn-B on the PCjr when this appears. Another DOS message asks if you want to terminate the batch job. Answer yes. The backup process is now complete. (Table 2 shows the screen messages and responses that should appear during this phase.)

# Table 2. End of Copy Phase

Screen Message:
REM \*\*\* COPY COMPLETE
A>ERASE FILECOPY.BAT
Insert disk with batch file
and press any key when ready
A>^C
Terminate batch job (Y/N)?

Operation/Response: Copies completed. Batch file erases itself.

and press any key when ready DOS error message. Press any key A>^C Press BREAK.

Terminate batch job (Y/N)? DOS message. Type Y.

Backup process finished; returned to DOS.

# **Additional Tips**

If you have another BASIC program in memory before running Filecopy, remember to save it on disk. Otherwise, it will be replaced when you load and run Filecopy.

Filecopy works with all versions of DOS, but subdirectories are supported only in DOS 2.1 or higher. Don't specify directory changes in the target or source paths if you're using an earlier version of DOS.

If you have two drives and generally use the first drive for the DOS disk and the second for your programming disk, change the first statement in line 100 from P\$="A:" to P\$="B:". This lets you keep the source disk in drive B and put the target disk in drive A.

# **IBM Filecopy**

For error-free program entry, be sure to use "The Automatic Proofreader," Appendix B.

```
KG 10 SCREEN 0:WIDTH 80:COLOR 7.0
Of 20 CLS:PRINT "Insert source disk in drive A an
     d hit any key to continue"
AH 30 GOSUB 700: A$=INPUT$(1)
MM 40 FILES
KD 50 PRINT: INPUT "Enter directory change (N=None
        > ",DIR$
CB 60 IF DIR$="N" OR DIR$="n" THEN 100
CH 70 ON ERROR GOTO 720
DK 80 CHDIR DIR$
NP 9Ø ON ERROR GOTO Ø:GOTO 4Ø
DL 100 P$="A:":GOSUB 420:CLS
LF 110 '*** get target path ***
60 120 INPUT "Target path" ,PATH$
DA 13Ø IF PATH$="B:" OR PATH$="b:" OR PATH$="A:"
      OR PATH$="a:" OR PATH$="" THEN 190
```

```
CF 140 PRINT: INPUT "Do you want to create this di
      rectory on the target disk (Y/N)", A$: IF A$
      ="N" OR A$="n" THEN 190 ELSE IF A$<>"Y" AN
      D A$<>"y" THEN 140
FI 15Ø ON ERROR GOTO 73Ø
F8 160 MKDIR PATH$:PRINT:PRINT:PRINT
P 170 '*** get files to copy ***
ON 180 ON ERROR GOTO 0
CK 190 FOR Z=0 TO FILENUM
EE 200 PRINT "COPY> ";FILE$(Z);TAB(24);"?";:COLO
      R 31,0
HN 210 GOSUB 700
M 220 A$=INPUT$(1):IF A$<>"Y" AND A$<>"y" AND A$
      <>"N" AND A$<>"n" THEN 220
LC 23Ø PRINT SPC(2);A$:COLOR 7,Ø
W 240 IF AS="Y" OR AS="y" THEN TY(Z)=1 ELSE TY(Z
      ) = \emptyset
IN 250 NEXT Z
60 260 INPUT "Is this all okay (Y/N) ";A$
05 270 IF A$="N" OR A$="n" THEN CLS:GOTO 190
00 280 IF A$="Y" AND A$<>"y" THEN 260
LC 290 '*** batch file creation ***
PF 300 OPEN "FILECOPY.BAT" FOR OUTPUT AS #1
GJ 31Ø PRINT #1, "VERIFY ON"
80 320 FOR Z=0 TO FILENUM
IH 330 IF TY(Z)=0 THEN 350
HO 340 PRINT #1, "COPY "; FILE$(Z); " "; PATH$
IN 350 NEXT Z
00 36Ø PRINT #1, "REM *** COPY COMPLETE"
66 370 PRINT #1, "ERASE FILECOPY.BAT"
AF 380 CLOSE #1
HD 390 '*** enter DOS ***
HN 400 CLS: SYSTEM
AA 410 '*** directory read in array FILE$ ***
IK 420 DEF SEG=0
OK 430 CLS:COLOR 31,0:PRINT"One moment please"
LA 440 COLOR 7,0:ON ERROR GOTO 460
AD 450 FILES P$: ON ERROR GOTO 0:GOTO 470
HN 460 BEEP: COLOR 31:CLS:PRINT"Cannot read direct
      ory": COLOR 7: ON ERROR GOTO Ø: END
BD 470 DIM TEM$(48):LOCATE 3,1:COLOR 7:ROWS=0
LL 480 POKE 1050,30:POKE 1052,34:POKE 1054,0:POKE
       1055,79: POKE 1056,13: POKE 1057,28
5 490 LINE INPUT TEM$ (ROWS)
PS 500 IF TEM$(ROWS)<>"" THEN ROWS=ROWS+1:60TO 48
      Ø
E 51Ø DIM FILE$(ROWS*4-1), TY(ROWS*4-1)
HE 520 ROWS=ROWS-1
SJ 53Ø FOR Z=Ø TO ROWS
```

FC 54Ø FOR Z1=Ø TO 3

- Of 600° \*\*\*\* remove spaces from filename \*\*\*

  BP 610 FOR Z=0 TO FILENUM
- CK 62Ø A\$="":PERIOD=Ø
- CJ 63Ø FOR Z1=1 TO 17
- 00 640 IF MID\$(FILE\$(Z),Z1,1)=" " THEN 660 ELSE I F MID\$(FILE\$(Z),Z1,1)=". " THEN PERIOD =1
- LN 650 A\$=A\$+MID\$(FILE\$(Z), Z1, 1)
- KN 660 NEXT Z1
- EB 670 FILE\$(Z)=A\$: IF PERIOD=0 THEN FILE\$(Z)=FILE \$(Z)+"."
- EO 680 NEXT Z:RETURN
- DD 690 \*\*\*\* clear keyboard buffer \*\*\*
- FL 700 DEF SEG=0:POKE 1050, PEEK (1052):RETURN
- OH 710 '\*\*\* error messages \*\*\*
- M 72Ø BEEP: COLOR 31, Ø: PRINT "Directory does not exist": COLOR 7, Ø: RESUME 5Ø
- MM 730 BEEP:COLOR 31,0:PRINT:PRINT:PRINT "Cannot create directory -- reenter path":COLOR 7, 0:RESUME 120

# Beginner's Guide to Typing In Programs

The programs for the IBM PC and PCjr published in this book are written in a language known as BASIC. BASIC (Beginner's All-purpose Symbolic Instruction Code) is the most common programming language for home and personal computers. The programs in this book are BASIC listings generated on a printer. By typing the listing *exactly* as it appears and then saving it on disk, you can add the program to your software library.

If you've never typed in a program listing, please observe

the following precautions to avoid problems.

First of all, be sure the program listing is the correct one for your computer. Though most of the programs in this book work as printed on both the PC and PCjr, a few work only on the PCjr or only on the PC.

Second, be sure the program will work on your system configuration. The programs in this book will work only on specific configurations. Special requirements can usually be found at the beginning of the articles. The most common requirement is the color/graphics adapter for the IBM PC.

Third, notice which version of BASIC is required. There are several versions of BASIC for IBM personal computers. These versions are upward-compatible with each other—meaning that programs written in the lower versions will work

in the upper versions, but not necessarily vice versa.

Both the PC and PCjr have a built-in version of BASIC called Cassette BASIC. This is the simplest BASIC; programs written in Cassette BASIC will work with all other BASICs. The computer automatically enters Cassette BASIC whenever you switch on the machine without a DOS disk in the disk drive (or if you have no disk drive). As the name implies, Cassette BASIC is designed for using tape storage instead of disk storage.

The next higher version is Disk BASIC. It is included on your DOS disk. To load Disk BASIC on a PC, switch on the computer with the DOS disk inserted in the drive (drive A on dual-drive systems). At the DOS prompt A>, type BASIC and press the Enter key.

The next higher version on the PC is called Advanced BASIC, or BASICA. Load BASICA the same way you would load Disk BASIC, except type *BASICA* at the DOS prompt. Like Disk BASIC, BASICA is included on your DOS disk.

A slightly higher version of BASIC is available for the PCjr: Cartridge BASIC. This is a plug-in ROM cartridge available from IBM at extra cost. Cartridge BASIC is BASICA, but has extra commands to take advantage of the PCjr's special features. Just insert Cartridge BASIC and type BASIC at the DOS prompt, then press Enter.

# **Computers Are Picky**

Unlike the English language, which is full of ambiguities, BASIC usually has only one "right way" of stating something. Each letter, symbol, and number is significant. Type the listings exactly as they appear. A common mistake is to substitute the letter O for the numeral 0 or a lowercase l for the numeral 1. Also, all punctuation must be entered as it appears in the listing. Substituting a comma for a semicolon or omitting a colon can make a big difference. Even blank spaces can be important.

## **DATA Statements**

Some programs may have one or more sections of DATA statements. These lines are especially critical.

If a single number in a DATA statement is mistyped, your computer could lock up, or crash—the screen may go blank and the keyboard may refuse to acknowledge any more commands. The computer cannot be harmed by this. However, to regain control, you may have to reset the machine by pressing Ctrl-Alt-Del or by turning the power switch off and on again. Both actions will erase the BASIC program.

For this reason, always save a copy of the program before running it for the first time. If the computer crashes, you can reload the program and check for the error. It's also a good idea to save the program periodically as you type it in.

**Know Your Computer** 

You should familiarize yourself with your computer before trying to type in a program. Learn the commands for saving and loading programs. Learn to use the keyboard editing functions. You could retype a line if you make a mistake, but it's much easier to insert and delete characters with the Ins and Del keys.

# The Automatic Proofreader

Charles Brannon

Now there's a way to banish practically all typing errors when entering programs in this book. "The Automatic Proofreader" instantly checks your typing as you enter each line. The Proofreader works on any IBM PC or Enhanced Model PCjr with Cartridge BASIC.

We all know it's hard to type in a program correctly the first time. Seemingly trivial typing errors can trigger dreaded error messages or even a *system crash* (the computer locks up and the keyboard won't respond). Usually, the only way to recover from such a crash is to reset the computer by turning it off, then on again—wiping out the memory and all your typing in the process.

Even when you locate and correct the mistyped lines, there always seem to be more, lurking somewhere in the hundred-odd lines of the program. Sometimes you feel like giving up.

#### **Elusive Errors**

Some errors are almost impossible to catch, especially if you know little or nothing about BASIC programming. For instance, can you spot the mistake in this line?

100 C = C + LEN(STR\$(VAL(L\$)) + 1

Here's how it should read:

100 C = C + LEN(STR\$(VAL(L\$))) + 1

Did you catch the difference? A right parenthesis was missing before the  $\pm 1$ . (A left parenthesis must always have a matching right parenthesis. If you add up all the parentheses in a statement, you should get an even number.)

An Impossible Dream?

The strong point of computers is that they excel at tedious, exacting tasks. So why not get your computer to check your typing for you? An impossible dream?

Not any more—not with "The Automatic Proofreader." The Automatic Proofreader is a BASIC program that mimics the IBM BASIC program editor, although it's a little slower. In effect, with the Proofreader you'll be using a BASIC program to enter other BASIC programs.

The Proofreader lets you enter program lines as you normally do, but with an important difference. After you type in a line and press Enter, a pair of letters appears, inserted just before the line you've typed. This pair of letters is called a checksum. You compare the checksum to a matching code of letters in the program listing. If the pair of letters on your screen matches the pair of letters in the program listing, the line was entered correctly. A glance is all it takes to confirm that you've typed the line right.

Does it sound too good to be true? It isn't. Thousands of readers of our books, COMPUTE! magazine, COMPUTE!'s Gazette, and COMPUTE!'s Apple Applications Special issues have been successfully using similar Proofreaders to type in program listings for their Commodore, Atari, Apple, and IBM

computers.

Using the Proofreader

To get started, type in the Automatic Proofreader listing at the end of this appendix and save a couple of copies. You'll want to use it whenever you enter a program from this book, COM-PUTE! magazine, and other COMPUTE! books. Naturally, the Proofreader can't check itself, so you'll have to be extra careful when typing it in. Often, when readers experience difficulty with the Proofreader, the problem has been traced to improperly typing the Proofreader program. So check every line very carefully—if you get it right, it'll probably be the last program listing in this book you'll ever have to scrutinize that closely.

When you run the Automatic Proofreader, the screen clears to white and the prompt *Proofreader Ready* appears. At this point, the Proofreader is ready to accept program lines or commands. You can just type in a program as you normally

would.

Here's an example of how it works. Type in the following line:

**120 RESUME 130** 

When you press Enter, there'll be a short delay and the checksum will appear:

#### **BE 120 RESUME 130**

The two letters *BE* are the checksum. Try making a change in the line, then press Enter. Notice that the checksum has changed. The slightest alteration to the line results in a different checksum.

All the program listings published in this book have a checksum printed to the left of each line number. Just type in each line (omitting the printed checksum, of course), and compare the checksum on your screen with the checksum in the listing. If they match, go on to the next line. If they don't match, there's a difference between the way you typed the line and the way it appears in the book. It might be a very slight difference that's hard to spot at first. When you find it, you can correct the line immediately with the cursor and editing keys instead of waiting to find the error when you run the program.

Although the Proofreader is an indispensable aid, there are a few things to watch out for. First, the Proofreader is *very* literal: it looks at the individual characters in a line. It makes a distinction between upper- and lowercase, so be sure to *leave Caps Lock on* while you type in a listing, releasing it when necessary to enter lowercase. For similar reasons, do not use? as an abbreviation for PRINT—they're not the same thing in the Proofreader's eyes. Neither are the numeral 1 and the lowercase *l* and the uppercase *l*, or the numeral 0 and the uppercase *O*. The Proofreader can even catch transposition errors—such as PIRNT instead of PRINT.

The Proofreader is also picky with spaces, since proper spacing is important to prevent syntax errors in IBM BASIC. Adding an extra space or leaving one out—even in places where it's normally permitted, such as within PRINT statements or REMarks—will result in a different checksum. If you want to modify something, we recommend first typing the program exactly as published and verifying that it runs, then making your modifications.

#### **Proofreader Commands**

The Proofreader has many commands, almost identical in syntax to those found in IBM BASIC. In fact, the editing environ-

ment is so similar that you may forget you're using a BASIC program to enter other BASIC programs. If in doubt, remember that BASIC's prompt is OK, while the Proofreader says *Proofreader Ready*. Also, the screen is white when the Proofreader is active.

LIST works just the way it does in BASIC. LIST 10 lists just line 10. LIST 40–90 displays all lines between 40 and 90, inclusive. LIST 100– gives you all the lines from line 100 to the end of your program. If you press any key while the program is LISTing, the listing will stop. Unless you are running the Proofreader under PCjr Cartridge BASIC or BASICA 2.0 or 2.1, do not press Ctrl-Break to stop the listing or you will exit the Proofreader. The Break key is trapped with Advanced BASIC 2.0 or 2.1, so you'll get the message *Stopped*.

CHECK is a special Proofreader command that acts like LIST, except it also displays the checksum for each line.

LLIST will list the program to the current printer device. It works as LLIST does in BASIC.

NEW clears out the program in memory—not the Proof-reader, but the program you're typing. However, there's an extra safeguard built in. Unlike BASIC, the Proofreader will ask, *Erase program—Are you sure?* You must enter Y to erase the program. Remember, this won't remove the Proofreader itself, but only the program held by the Proofreader.

FILES lists the disk directory on the screen. It lists only

the directory for drive A.

BASIC exits the Proofreader. It returns you to BASIC's OK prompt and returns the screen color to black, leaving the Proofreader still in memory. To be safe, always save your program on disk before leaving the Proofreader. If you accidentally exit the Proofreader by typing BASIC, you can reenter the Proofreader and retrieve your program by typing CONT. You'll get a syntax error message, and the screen won't return to white, but the program you were typing will be intact.

#### SAVE and LOAD Commands

You can save a program at any point when using the Proof-reader. In fact, it's a good idea to save a program occasionally as you're typing so that an unexpected power failure won't wipe out all your work. Just type SAVE "filename". As usual, the ending quotation mark is optional. If you don't enter a period and a three-character extender, the extender .BAS will be

automatically appended. Again, this is just like IBM BASIC.

Unlike IBM BASIC, the Proofreader always saves programs to disk in ASCII form. You can load this program from BASIC like any other. Since ASCII files take up more room on a disk than ordinary program files, later you may want to resave the program back to disk from BASIC in order to conserve disk space.

You can reload programs into the Proofreader with the command LOAD "filename". (As with the SAVE command, the extender .BAS is assumed if you don't enter an extension.) This way, you can type in part of a long program, save it on disk, and load it again later to continue typing. But make sure the program you're loading was saved by the Proofreader. The Proofreader cannot successfully load a program file that's not in ASCII form.

Checksum programs are not new in computer books and magazines. But unlike other checksum programs, The Automatic Proofreader shows you *instantly*, as soon as you've entered the line, if you made a typo. We hope that the Proofreader makes your program entry both faster and easier, and that you'll never have to face another frustrating error message again.

#### The Automatic Proofreader

- LA 10 'Automatic Proofreader Version 2.00 (Lines 270,510,515,517,620,630 changed from V1.0)
- U 100 DIM L\$(500), LNUM(500): COLOR 0,7,7: KEY OFF: CLS: MAX=0: LNUM(0)=65536!
- PK 110 ON ERROR GOTO 120:KEY 15, CHR\$(4)+CHR\$(70): ON KEY(15) GOSUB 640:KEY (15) ON:GOTO 130
- BE 120 RESUME 130
- N 130 DEF SEG=&H40: W=PEEK (&H4A)
- IH 140 ON ERROR GOTO 650:PRINT:PRINT"Proofreader Ready."
- KB 15Ø LINE INPUT L\$:Y=CSRLIN-INT(LEN(L\$)/W)-1:LO CATE Y,1
- CA 160 DEF SEG=0:POKE 1050,30:POKE 1052,34:POKE 1 054,0:POKE 1055,79:POKE 1056,13:POKE 1057, 28:LINE INPUT L\$:DEF SEG:IF L\$="" THEN 150
- 8C 17Ø IF LEFT\$(L\$,1)=" " THEN L\$=MID\$(L\$,2):60TD
  17Ø
- NN 18Ø IF VAL(LEFT\$(L\$,2))=Ø AND MID\$(L\$,3,1)=" "
  THEN L\$=MID\$(L\$,4)
- A6 19Ø REM

- ND 200 IF ASC(L\$)>57 THEN 260 'no line number, the erefore command
- JB 205 BL=INSTR(L\$," "):IF BL=0 THEN BL\$=L\$:GOTO 206 ELSE BL\$=LEFT\$(L\$,BL-1)
- GH 206 LNUM=VAL(BL\$):TEXT\$=MID\$(L\$,LEN(STR\$(LNUM)
  )+1)
- 06 210 IF TEXT\$="" THEN GOSUB 540:IF LNUM=LNUM(P) THEN GOSUB 560:GOTO 150 ELSE 150
- ## 220 CKSUM=0:FOR I=1 TO LEN(L\$):CKSUM=(CKSUM+AS C(MID\$(L\$,I))\*I) AND 255:NEXT:LOCATE Y,1:P RINT CHR\$(65+CKSUM/16)+CHR\$(65+(CKSUM AND 15))+" "+L\$
- JE 230 GOSUB 540: IF LNUM(P)=LNUM THEN L\$(P)=TEXT\$
  :GOTO 150 'replace line
- CL 240 GOSUB 580:GOTO 150 'insert the line
- AD 260 TEXT\$="":FOR I=1 TO LEN(L\$):A=ASC(MID\$(L\$, I)):TEXT\$=TEXT\$+CHR\$(A+32\*(A>96 AND A<123)):NEXT
- P 270 DELIMITER=INSTR(TEXT\$," "):COMMAND\$=TEXT\$:
  ARG\$="":IF DELIMITER THEN COMMAND\$=LEFT\$(T
  EXT\$,DELIMITER-1):ARG\$=MID\$(TEXT\$,DELIMITE
  R+1) ELSE DELIMITER=INSTR(TEXT\$,CHR\$(34)):
  IF DELIMITER THEN COMMAND\$=LEFT\$(TEXT\$,DEL
  IMITER-1):ARG\$=MID\$(TEXT\$,DELIMITER)
- FC 28Ø IF COMMAND\$<>"LIST" THEN 41Ø
- 10 290 OPEN "scrn:" FOR OUTPUT AS #1
- LH 300 IF ARG\$="" THEN FIRST=0:P=MAX-1:GOTO 340
- I) 310 DELIMITER=INSTR(ARG\$,"-"):IF DELIMITER=0 T HEN LNUM=VAL(ARG\$):GOSUB 540:FIRST=P:GOTO
- EC 330 LNUM=FIRST:GOSUB 540:FIRST=P:LNUM=LAST:GOS UB 540:IF P=0 THEN P=MAX-1
- 60 340 FOR X=FIRST TO P:N\$=MID\$(STR\$(LNUM(X)),2)+
- KA 35Ø IF CKFLAG=Ø THEN A\$="":GOTO 37Ø
- Ff 360 CKSUM=0:A\$=N\$+L\$(X):FOR I=1 TO LEN(A\$):CKS
  UM=(CKSUM+ASC(MID\$(A\$,I))\*I) AND 255:NEXT:
  A\$=CHR\$(65+CKSUM/16)+CHR\$(65+(CKSUM AND 15))+" "
- 00 37Ø PRINT #1, A\$+N\$+L\$(X)
- JJ 380 IF INKEY\$<>"" THEN X=P
- OF 390 NEXT :CLOSE #1:CKFLAG=0
- CA 400 GOTO 130
- PD 410 IF COMMAND\$="LLIST" THEN OPEN "lpt1:" FOR OUTPUT AS #1:GOTO 300
- 6H 42Ø IF COMMAND\$="CHECK" THEN CKFLAG=1:GOTO 29Ø
- KA 43Ø IF COMMAND\$<>"SAVE" THEN 45Ø

- CL 440 GOSUB 600:OPEN ARG\$ FOR OUTPUT AS #1:ARG\$=
  "":GOTO 300
- DE 450 IF COMMAND\$<>"LOAD" THEN 490
- P6 460 GOSUB 600:OPEN ARG\$ FOR INPUT AS #1:MAX=0: P=0
- KA 470 WHILE NOT EOF(1):LINE INPUT #1,L\$:BL=INSTR
   (L\$," "):BL\$=LEFT\$(L\$,BL-1):LNUM(P)=VAL(BL
   \$):L\$(P)=MID\$(L\$,LEN(STR\$(VAL(BL\$)))+1):P=
   P+1:WEND
- KK 48Ø MAX=P:CLOSE #1:GOTO 13Ø
- 06 49Ø IF COMMAND\$="NEW" THEN INPUT "Erase progra
  m -- Are you sure";L\$:IF LEFT\$(L\$,1)="y" OR
   LEFT\$(L\$,1)="Y" THEN MAX=Ø:GOTO 13Ø:ELSE
  13Ø
- CL 500 IF COMMAND\$="BASIC" THEN COLOR 7,0,0:ON ER ROR GOTO 0:CLS:END
- NC 510 IF COMMAND\$<>"FILES" THEN 520
- IN 515 IF ARG\$="" THEN ARG\$="A:" ELSE SEL=1:GOSUB 600
- 10 517 FILES ARG\$: GOTO 130
- M 520 PRINT"Syntax error": GOTO 130
- B0 540 P=0:WHILE LNUM>LNUM(P) AND P<MAX:P=P+1:WEN
  D:RETURN</pre>
- KM 560 MAX=MAX-1:FOR X=P TO MAX:LNUM(X)=LNUM(X+1)
  :L\$(X)=L\$(X+1):NEXT:RETURN
- 6K 58Ø MAX=MAX+1:FOR X=MAX TO P+1 STEP -1:LNUM(X)
  =LNUM(X-1):L\$(X)=L\$(X-1):NEXT:L\$(P)=TEXT\$:
  LNUM(P)=LNUM:RETURN
- 8A 600 IF LEFT\*(ARG\*,1)<>CHR\*(34) THEN 520 ELSE A RG\*=MID\*(ARG\*,2)
- EE 610 IF RIGHT\$(ARG\$,1)=CHR\$(34) THEN ARG\$=LEFT\$
  (ARG\$,LEN(ARG\$)-1)
- LA 620 IF SEL=0 AND INSTR(ARG\$,".")=0 THEN ARG\$=A RG\$+".BAS"
- 00 630 SEL=0: RETURN
- HM 640 CLOSE #1:CKFLAG=0:PRINT"Stopped.":RETURN 1
- II 650 PRINT "Error #"; ERR: RESUME 150

## **Disk Instructions**

Typing in long BASIC programs can be a time-consuming task. As a service to our readers, COMPUTE! Publications has made available for purchase a disk which contains all the programs in this book. If you wish to buy *COMPUTE!'s Second Book of IBM* disk, use the coupon found in the back of this book, or call toll-free 1-800-346-6767 (in NY, 212-887-8525).

Preparing a Purchased Disk

If you've purchased *COMPUTE!'s Second Book of IBM* disk and simply try to insert it in your drive and turn on the computer, you'll get the following message:

Non-System disk or disk error

The disk you purchased doesn't contain the IBM system files necessary to start the IBM. You have two options. You can boot the system each time with a disk that *does* contain COMMAND.COM and BASICA (BASICA is not necessary with the PCjr), or you can copy the system files to your program disk by following the directions below for your system.

Single-Drive Systems

- 1. As a precaution, cover the notch on the PC-DOS master disk (or a copy of the master disk) with one of the write-protection tabs that come with each box of disks. This will prevent you from accidentally writing data to the wrong disk.
- 2. Turn on your IBM with a PC-DOS master disk inserted in the disk drive.
- 3. Enter this command at the A> prompt:
  - SYS B: (and press Enter)
- You'll have to swap disks a few times. You should insert the PC-DOS master disk in the disk drive whenever you see the message

Insert diskette in drive A: and strike any key when ready

And insert the COMPUTE!'s Second Book of IBM disk whenever you see the message

### Insert diskette in drive B: and strike any key when ready

Be sure you insert the correct disk each time.

5. Enter the following command at the A> prompt, swapping disks as necessary:

#### COPY COMMAND.COM B: (and press Enter)

6. If you're using a PC, you'll also need to copy BASICA or BASIC. After entering the following command at the A> prompt, insert a disk that has BASICA or BASIC on it when prompted for drive A, and insert the COMPUTE!'s Second Book of IBM disk when prompted for drive B.

#### COPY BASICA.COM B:

or

#### COPY BASIC.COM B:

You're all ready to use the disk.

#### **Double-Drive Systems**

- 1. As a precaution, cover the notch on the PC-DOS master disk (or a copy of the master disk) with one of the write-protection tabs that come with each box of disks. This will prevent you from accidentally writing data to the wrong disk.
- 2. Turn on your IBM with a PC-DOS master disk inserted in disk drive A and the COMPUTE!'s Second Book of IBM disk in drive B.
- 3. Enter this command at the A> prompt:

SYS B: (and press Enter)

4. Enter this command at the A> prompt:

COPY COMMAND.COM B: (and press Enter)

5. If you're using a PC, you'll also need to copy BASICA or BASIC. Insert a disk that has BASICA or BASIC on it in drive A. Enter this at the A> prompt:

#### **COPY BASICA.COM B:**

or

#### **COPY BASIC.COM B:**

You're all ready to use the disk.

Using the COMPUTE!'s Second Book of IBM Disk
Once you've added the system files, your disk will load and
run automatically. Insert the COMPUTE!'s Second Book of IBM
disk in drive A. If your computer is still on, hold down the Alt
and Ctrl keys while pressing the Del key; if your computer is
off, turn it on. Once the system is booted, follow the screen
instructions to select the program you wish to run.

#### Preparing a Blank Disk

**Warning:** The instructions listed below are **not** for use with the *COMPUTE!'s Second Book of IBM* disk that can be purchased from COMPUTE! Publications. If you have purchased a disk, see the instructions that begin at the heading "Preparing a Purchased Disk" above.

If you've chosen to type in the programs, the first step is to format a disk. You'll need a blank double-sided, double-density, 5¼-inch floppy disk. (Although single-sided disks are also acceptable, we recommend double-sided. The Format command listed below is for double-sided disks.) Be sure that you use a blank disk since formatting a disk erases any data that might be on the disk. Insert your PC-DOS master disk (or a copy of the master disk) into drive A. Enter the date and time when asked.

At the A> prompt enter the following command:

FORMAT B:/S/V (and press Enter)

The drive will spin and then you'll see

Insert new diskette for drive B: and strike any key when ready

Now, insert the blank disk in drive B if you have a dual-drive system, or replace the DOS disk in drive A with a blank disk if you have a single-drive system. Press any key. When the formatting is complete, you'll be prompted for a volume name. Enter a name for this disk.

Since most of the programs in this book are written in BASIC, we suggest that you copy BASICA.COM (or BASIC.COM) to this disk (copying BASICA is not necessary for the PCjr since the Enhanced PCjr has BASIC on a cartridge). Insert a disk containing BASICA.COM into drive A

and your newly formatted disk in drive B (if you use a single-drive system, you'll be prompted to insert the new disk). Enter the following at the A> prompt:

COPY BASICA.COM B: (and press Enter)

01

#### COPY BASIC.COM B:

To begin typing in a BASIC program, enter

BASICA (and press Enter)

You're now ready to type in and save any of the programs in COMPUTE!'s Second Book of IBM. We suggest you read Appendices A and B, and type in and save "The Automatic Proof-reader" first.

## Index

chain 210 \$ See dollar sign % See dummy parameter character generator 35 CHECK 285 = See equality symbol - See hyphen checksum 284-86 Chime (/C) 28 CHKDSK 209 <> See inequality symbol \* See pattern-matching symbols CHOOSE 215, 220 "CHOOSE.COM Filemaker" program 221 ? See pattern-matching symbols / See slash CHR\$ 237 A> (DOS prompt) 201 Advanced BASIC 115, 280. See also BASICA circle 173 CIRCLE 193 CLEAR 115 AH register 203 alphanumerics 214 altitude, sun 124 Alt key 265 AND 23, 194 code segment register 202, 204 colon (:) 212, 264, 274 COLOR 193 Angell, Ian O. 179
"Animator" program 144-56 color bits table 159 "Color Directory" program 7-10 color graphics 193 ANSI.SYS driver 219, 220 apostrophe 264 color programs descriptions 164 Applied Concepts in Microcomputer Graphics comma 23, 268 COMMAND /C 210 179 compiler BASIC 234 array 265 COMPUTE!'s Second Book of IBM disk ARRAY 194 preparing 289-90 array, temporary 7 using 291-92 array variables 255, 259 arrow key 265 condensed mode 5 "Conical Helix" program 184–85 CONNECT 237 "ARRYSCRN Source Code" program 273 artifacting 161 CONT 265, 266 Artwick, Bruce A. 179 ascension, sun 124 coordinate system translation 176 COPY 5, 212, 274 ASCII 6 asterisks 268 COPY CON: 210 cosine 42 asynchronous 236 **AÚTO 264** Crosstalk 230 Ctrl-Alt-Del keys 216, 219, 265 AUTOEXEC.BAT 209, 215 "Automatic Proofreader, The" program Ctrl-Break keys 214 286 - 88Ctrl key 265 curves, fractal 186-90 backup 274 curves, plotting 178 D. See Dump "Balloon Crazy" program 79–83 BASIC 3, 279, 285 BASICA 115, 193, 280. See also Advanced DATA 23, 28, 280 database, searching 22 BASIC BASIC compiler 241 data file 11 Basic Input/Output System. See BIOS data segment (DS) 204 DATE 209, 215 batch program 209-20 "Bearmath" program 116-21 binary code 205-6 "Deadly Dungeon" 92-99 DEBUG utility 201-8 binary to decimal conversion table 160 BIOS (Basic Input/Output System) 27, 35 backing up 202 copying 202 bit 158 debugging 206 prompt 201 bit pattern 159 BLOAD 227 running 206 Boolean variable 14, 16 declination, sun 124 DELAY! 196 braces 217 brackets 218 delimiter 203 BREAK ON 220 description file 4 destination label 216 "Buffer Finder for Compatibles" program 252 byte 157 DIMension 265 DIR 3, 267 /C. See Chime CALL 227 disassembler 201 disc command 233 Cartridge BASIC 193, 265, 280 Cassette BASIC 267, 279 disk, COMPUTE!'s Second Book of IBM preparing 289-90 using 291-92 celestial equator 125

D: 1 P1010 000	
Disk BASIC 280	geocentric angle 123
DISKCOPY 274	GET 194
disk drive 212	GOTO 211, 212, 216, 255
source 212	
	graphics, color 193
target 212	graphics images, storing 157
DJSYMBOL 235	graphics mode 35
dollar sign 203, 268	guard time 239
DOS 265	HEAD 6
disk 279	
	"Hickory Dickory Dock" program 135–38
function routine 203	hidden-surface elimination 178
prompt ( A>) 201	hyphen 201. See also DEBUG prompt
DOS Supplemental Programs disk 201	IBM BASIC 112
double-precision variable 259	"IBM Filecopy" program 276-78 "IBM Pie Chart Maker" program 43-48 "IBM Variable Snapshot" program 260-63
Day Jones News Coming 220 41	IDM Friedopy program 270-76
Dow Jones News Service 230–41	IBM Pie Chart Maker program 43–48
opening an account 241	"IBM Variable Snapshot" program 260-63
dragon sweep 187	IF 211-14, 216
DRAW 193	ifier 238
dropped by host system message 233	IN 213
dropped by nost system message 255	
dummy parameter 211	inequality symbol 113, 265
Dump (D) 207	initialize 265. See also formatting a disk
E. See Enter	INKEY\$ 112
	INPUT 112
ECHO 211, 220 ECHO OFF 211	
	INPUT\$ 237
ECHO ON 211	instruction pointer 204
editing, macro 143	INT 203, 205
"Eight Thousand Dragons" program 191 "80-Column Billboards" program 38–39	integer variable 227, 255, 259
"80-Column Billboards" program 38-39	intensity 158.
8088 microprocessor 204	
8088 microprocessor 204	interpreter BASIC 234
EDLIN program 210, 215	"Jackpot" program 73–77
electron guns 158	"Jackpot" program 73–77 "Jigsaw" program 87–88 KEY 267
elongation, moon 124	KEY 267
Enter (E) 207	keyboard buffer 6
ENTER QUERY 233	and the second s
	kilometers, calculating 124
enter terminal identifier message 237	L. See Load
enter user name: message 238	latitude 123
ENTRIES 7	leap day 14
EOF 236	letters, checking for 113
equality symbol 214	"Lightning Sort Loader" 228
equinox, vernal 113-14	LINE 162, 193, 194
ERASE 234	LINE INPUT 6
ERRORLEVEL 214, 216–17	
error message 4, 233	lines, flickering 35 LIST 264, 266, 285
EXIST 214	LIST 204, 200, 200
	LLIST 285
"Extending Definitions" program 253-54	Load (L) 202
extension 3	LOAD 266, 285, 286
extra segment (ES) 204 "Fast Filer" program 25–26	loading a program 266
"Fast Filer" program 25-26	LOC 236
FCB. See file control block	
	LOCATE 236
Fibonacci sequence 54	LPT1 267
file control block (FCB) 202	LSORT 227
FILECOPY 275	luminance 158
"FILEGRP.BAS" program 225	/M. See Military time
filename 3 265	macro editing 142
"FILES .BAT" program 221–23 "FILES.HLP" program 224 "FILES.MNU" program 223	macro editing 143
FILES 3, 0, 207, 203	Mandelbrot, Benoit 186
"FILES.BAT" program 221–23	"Memo Diary" program 16-21
"FILES.HLP" program 224	MENU 209
"FILES,MNU" program 223	Microsoft BASIC 112, 193
"40-Column Billboards" program 37–38	
For Property 1000	Military time (/M) 28
Fn-Break keys 265	"Million-Color Palette" programs 166-69
Fn-Pause keys 264	miniassembler 201
FOR 211–14	mnemonics 205
FORMAT 265	MODE 234
formatting a disk 265, 291	modem 235–36
four-color mode bit patterns translations 165	moon
fractal curves 186–90	age 124
fractional dimension 186	phase 124
function keys 250-52, 266	MOV 205
generator 186	multitasking 27
0	11141114011116 47

N (Name) 202	running a program 266
NEW 285	/S. See Standard time
NO CARRIER 231, 237	SAVE 266, 285
Norton Utility program 238	saving a program 266
NOT 214	scalar variables 255, 259
nybble 159	screen 115
ON 234	active 115
ON ERROR 234 OPEN 6	visual 115
operand 203, 205	"Screen Clock" program 30–34
destination 203	screen flipping 114 screen mode memory requirements 158
source 203	"Screen Swapping Routines" program 271
option 194	SCREEN 2 gray scales table 163
OPTION BASE 257–58	"SCRNARRY Source Code" program 272
OR 23, 194	segments 204
PAINT 42, 159, 163, 193	code segment (CS) 204
PALETTE 159, 161, 162	data segment (DS) 204
paragraphs 204	extra segment (ES) 204
parameters 210, 211, 219	stack segment (SS) 204
dummy 211	self-avoiding curves 186. See also curves,
pattern-matching symbols 213 pause 143	fractal
PAUSE 211–14	self-contacting curves 186. See also curves, fractal
PCcrayon program 195	self-similar curves 186. See also curves, fractal
PC-DOS 3, 27	semicolon 232, 238
PCOPY 115	set value 213
PC-Talk program 230-39	SHIFT 211, 212
PEL map 36	sine 42
phase, moon 124	single-precision variable 255, 259
phone, hanging up 239	single quotation mark 264
pie charts 40–42	"Skyscape" program 126-34 slash (/) 28, 233
pivot date 13 pixel 157–59, 170	"Slide Show Demo" program 272
pointers 6	"Snapshot Demo" program 263
polar coordinates 171	snowflake sweep 188
Practical Introduction to Computer Graphics, A	"Snowflake Sweep, The" program 192
179	soft key buffer 250-52
PRESET 194	"Sorting Demonstration" 229
PRICE 268	SOUND 196, 239
primary colors 158	source directory 274
PRINT 267	source disk 274
print string 203 PRINT USING 267	source drive 212
PSET 194	speed 143
pseudo-op. See pseudo-opcode	spiral 172–73 "Spiral Demo" program 271
pseudo-opcode 203	"Spiral Graphics" program 174-75
PUT 194	stack segment (SS) 204
Quit menu 5	Standard time (/S) 28
quotation mark 266	STOCK SYMBOL IN ERROR message 233
quotes, stock 233	string array 6
retrieving 238	string variable 259
"Quotes" program 241-49 /R. See Reset	subdirectory 274
radians 172	"Switchbox" program 57-61
raster beam 35	symbol file, creating 234 sysline 27–30
rectilinear coordinates, standard 171	SYSTEM 267
REM 211, 212, 218, 264	T. See Trace
RENUM 264	TAB 267
Reset (/R) 29	TAIL 6
"Restoring Function Definitions" programs	target drive 212
252–53	target path 274
RGB 158	telecommunications 230–41
right justification 267	displaying information 240
ROTANGL 172	ending 239
rotating a figure 172 "Rotating Graphics" program 173-74	receiving 236 sending 236
DI IN 266	tomporary array 7

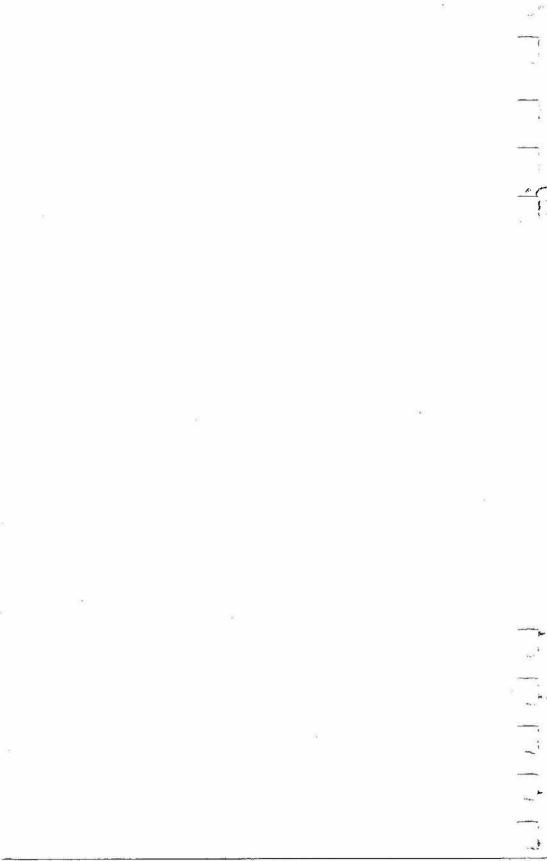
THEN 213
"3-D Graphics" program 181–83
three-dimensional plotting 176
tile painting 157, 159–63
TIME 28, 209, 215
time-delay loop 270
Trace (T) 207
TRACE 267
trapping keystrokes 112
TROFF 267
TRON 267
two-dimensional plotting 176
TYPE 220
"Type Bomb" 104–10
typing in programs 279–81, 291–92
U (Unassemble) 204, 205
/U (Update) 28
/V. See Volume label
variable names 264

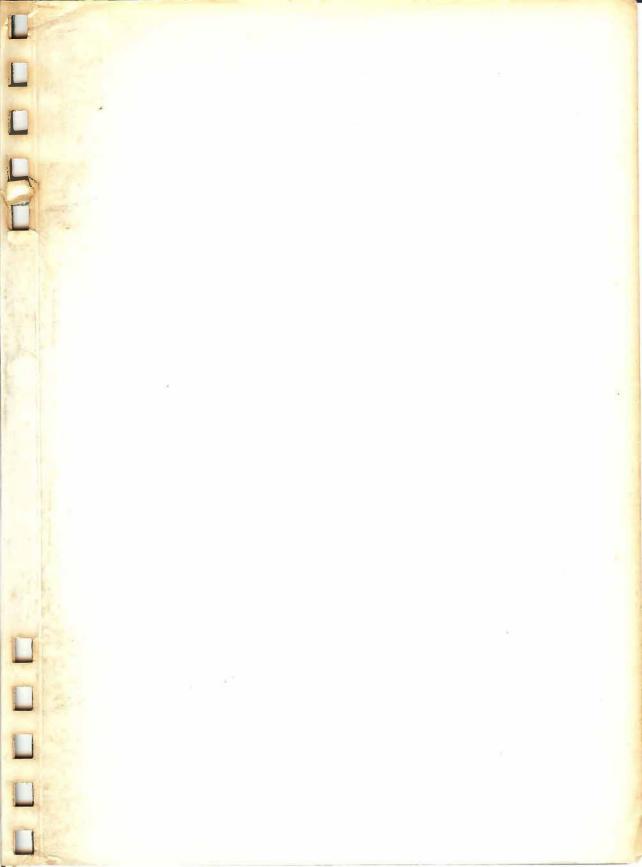
variables 255-59
vernal equinox 113-14
vertical retrace 35
video memory 35
Volume label (/V) 265
W. See Write
"Webster Dines Out" program 63-67
wildcards 6
"Witching Hour, The" program 69-71
word 205
Write (W) 202
x-axis 176
X-off code 239
XOR 194
y-axis 176
z-axis 176
z-axi

To order your copy of *COMPUTE!'s Second Book of IBM Disk*, call our toll-free US order line: 1-800-346-6767 (in NY 212-887-8525) or send your prepaid order to:

COMPUTE!'s Second Book of IBM Disk
COMPUTE! Publications
P.O. Box 5038
F.D.R. Station
New York, NY 10150

All orders must be prepaid (check, charge, or money order). NC residents add 4.5% sales tax.		
Send copies of <i>COMPUTE!'s Second Book of IBM Disk</i> at \$12.95 per copy. (467BDSK)		
Subtotal \$		
Shipping and Handling: \$2.00/disk \$		
Sales tax (if applicable) \$		
Total payment enclosed \$		
□ Payment enclosed □ Charge □ Visa □ MasterCard □ American Express		
Acct. No Exp. Date(Required)		
Name		
Address		
City State Zip		
Please allow 4-5 weeks for delivery.		





## IBM II

COMPUTE!'s Second Book of IBM has something for every member of the family. Programs and articles for the home, for the office, and for school provide instant power and practical applications for your IBM PC, PCjr, or PC-compatible personal computer.

Games teach and entertain; tutorials illuminate the mysteries of programming on the IBM; applications put your computer to work displaying messages, filing information, and remembering important dates; and graphics software animates figures, displays a million colors, and creates three-dimensional drawings.

COMPUTE!'s Second Book of IBM, with more than 30 programs and articles, includes:

- "Fast Filer," a simple-to-use program that electronically organizes almost any information.
- "Animator," a frame-by-frame animation program that lets you create short animation sequences.
- Thrilling games of strategy, like ``Switchbox'' and ``Deadly Dungeon.''
- Eductional software to teach math, typing, and telling time.
- "Skyscape," software that displays the night sky—from anywhere on the globe.
- Utilities to copy files quickly and sort through long lists in seconds.
- "Screen Swapping," a technique for rapidly flashing through graphics screens for presentations or "slide shows."
- And much more.

All the programs are ready to type in using "The Automatic Proofreader," an error-checking system which makes mistakes obsolete.

All the programs in this book are available on a companion disk. See the coupon in the back for details.